TEKNILLINEN KORKEAKOULU
Sähkö- ja tietoliikennetekniikan osasto

Kalle Koskinen

Käyttöliittymäkomponenttipohjainen web-sovelluskehitys

Diplomityö, joka on jätetty opinnäytteenä tarkastettavaksi diplomi-insinöörin tutkintoa varten Espoossa 6. syyskuuta 2005.

Työn valvoja Professori Heikki Saikkonen

Työn ohjaaja KTM Magnus Sjöblom

TEKNILLINEN KORKEAKOULU Diplomityön tiivistelmä

Tekijä: Kalle Koskinen

Työn nimi: Käyttöliittymäkomponenttipohjainen web-sovelluskehitys

Päivämäärä: 6. syyskuuta 2005 Sivumäärä: 82

Osasto: Sähkö- ja tietoliikennetekniikan osasto

Professuuri: T-106 Ohjelmistojärjestelmät

Työn valvoja: Professori Heikki Saikkonen

Työn ohjaaja: KTM Magnus Sjöblom

Tässä työssä tarkastellaan web-käyttöliittymien rakentamista käyttöliittymäkomponenteilla. Käyttöliittymäkomponentti sijoitetaan web-sivulle, jolloin se esittää tai vastaanottaa informaatiota käyttäjältä. Käyttöliittymäkomponentteja yhdistelemällä voidaan luoda monipuolisia ja käyttäjän valintoihin mukautuvia web-käyttöliittymiä.

Työssä tutustutaan ASP.NET- ja JavaServer Faces -käyttöliittymäkomponenttiteknologioihin, joita verrataan vanhempiin web-teknologioihin. Työn tarkoituksena on selvittää, mitä lisäarvoa komponenttiteknologiat tuovat web-sovelluskehitykseen, ja onko niillä mahdollista rakentaa suorituskykyinen web-sovellus.

Työn yhteydessä rakennettiin kaksi mahdollisimman identtistä käyttöliittymäkomponentteja hyödyntävää prototyyppiä, jotka vastaavat erästä tuotantokäytössä olevaa JavaServer Pages -pohjaista web-sovellusta. Prototyyppien käyttöliittymien rakentaminen sujui nopeasti ja vaivatta graafisia työkaluja käyttäen. Käyttöliittymäkomponentit helpottavat web-sivujen rakentamista merkittävästi, koska ne sisältävät paljon toiminnallisuutta, jonka toteuttaminen vanhemmissa web-teknologioissa on sovelluskehittäjän vastuulla. Suorituskykymittauksissa komponenttiteknologioilla tehdyt ASP.NET- ja JavaServer Faces -prototyypit pärjäsivät hyvin.

Avainsanat:

käyttöliittymäkomponentit, web-sovelluskehitys, ASP.NET, JavaServer Faces

HELSINKI UNIVERSITY OF TECHNOLOGY Abstract of Master's Thesis

Author: Kalle Koskinen

Title of thesis: GUI component-based web application development

Date: September 6, 2005 Number of pages: 82

Department: Electrical and Communications Engineering

Professorship: T-106 Software Technique

Supervisor: Professor Heikki Saikkonen

Instructor: Magnus Sjöblom, M.Sc. (Econ.)

This master's thesis concentrates on building web pages using graphical user interface (GUI) components. A GUI component is an entity that displays or receives information from a web user. Rich and adaptable user interfaces can be built by combining different GUI components.

This thesis focuses on component-based ASP.NET and JavaServer Faces web technologies, which are here compared with older web technologies. The purpose of this study is to investigate what added value GUI component-based technologies bring to the development of web applications. A secondary goal is to test whether or not the component technologies can be used to build an efficient web application.

Two almost identical GUI component-based prototypes were built in order to compare them with a JavaServer Pages-based reference web application. The user interfaces of the prototypes were built quickly and straightforwardly using integrated development environments. The study showed that the GUI components alleviate the complexity of web application development, since they provide a lot of functionality that previously needed to be implemented by the software developer. According to the executed performance tests, the ASP.NET and JavaServer Faces prototypes performed well.

Keywords: GUI components, web development, ASP.NET, JavaServer Faces

Alkulause

Tämä diplomityö on tehty kevään ja kesän 2005 aikana kustannuspaikkojen suunnittelu- ja raportointijärjestelmäprojektin osana Nokian Business Infrastructure -organisaatiossa.

Työn valvojaa, professori Heikki Saikkosta, haluan kiittää neuvoista ja ohjeista, jotka auttoivat erityisesti hahmottamaan ja jäsentämään tutkimusaluetta.

Työn ohjaajalle Magnus Sjöblomille kiitokset mahdollisuudesta tehdä työ mielenkiintoisesta aiheesta ja käyttää siihen riittävästi aikaa muista työkiireistä huolimatta. Kiitän myös lukuisista parannusehdotuksista, jotka selkeyttivät työn rakennetta ja tulosten esittelyä.

Lisäksi haluan kiittää Behzad Tabrizizadehia ripeästä mittausympäristöjen hankinnasta ja niiden pystyttämisessä saamastani avusta.

Aivan lopuksi haluan osoittaa kiitokset vanhemmilleni Karille ja Pirjolle tuesta ja kannustuksesta koko opintojeni ajalta.

Espoossa 6. syyskuuta 2005

Kalle Koskinen

Kalle Koskinen

Sisällysluettelo

Lyhenneluettelo

.NET	Microsoftin sovelluskehys.
ASP	Active Server Pages, Microsoftin web-teknologia.
BMP	Bitmap, eräs Microsoftin Windows-käyttöjärjestelmän kuvatiedostoformaatti.
C#	Microsoftin olio-ohjelmointikieli.
CGI	Common Gateway Interface, rajapinta web-palvelimen ja sen prosessin ulkopuolella ajettavan ohjelman välillä.
CLR	Common Language Runtime, ympäristö, joka suorittaa Microsoftin .NET-ohjelmia.
COM	Component Object Model, Microsoftin komponenttiteknologia.
CSS	Cascading Style Sheets, kieli, jolla voidaan liittää tyylimäärityksiä HTML-elementteihin.
DLL	Dynamic Link Library, dynaaminen aliohjelmakirjasto.
EJB	Enterprise JavaBeans, Java-teknologiaan perustuva hajautettujen järjestelmien arkkitehtuuri.
EL	Expression Language, ilmaisukieli.
ERP	Enterprise Resource Planning, toiminnanohjaus.
FCL	Framework Class Library, .NET-sovelluskehyksen luokkakirjasto.
FTP	File Transfer Protocol, Internetin tiedostojen siirtomenetelmä.
HTML	HyperText Markup Language, hypertekstin kuvauskieli.
HTTP	Hypertext Transfer Protocol, hypertekstin siirtoprotokolla.
IDE	Integrated Development Environment, integroitu kehitysympäristö.
IIS	Internet Information Server, Microsoftin web-palvelinohjelmisto.
ISAPI	Internet Server Application Programming Interface, Microsoftin web-palvelimen matalan tason ohjelmointirajapinta.
J2EE	Java 2 Platform Enterprise Edition, yrityskäyttöön tarkoitettu Java-alusta.
JDBC	Java Database Connectivity, Javan tietokantarajapinta.
JNDI	Java Naming and Directory Interface, Javan nimipalvelurajapinta.

JPEG	Joint Photographic Experts Group, suositun häviöllisen kuvanpakkausmenetelmän kehittänyt ryhmä.
JSF	JavaServer Faces, komponenttipohjainen web-teknologia.
JSP	JavaServer Pages, dokumenttipohjainen web-teknologia.
JVM	Java Virtual Machine, Java-virtuaalikone.
KP	Kustannuspaikka.
LAN	Local Area Network, lähiverkko.
MVC	Model-View-Controller, Malli-Näkymä-Ohjain-arkkitehtuurimalli.
NCSA	National Center for Supercomputing Applications, Yhdysvaltojen kansallinen laskentakeskus.
PHP	PHP: Hypertext Preprocessor, avoimeen lähdekoodiin perustuva web-teknologia.
PNG	Portable Network Graphics, ISO-standardoitu häviötön kuvatiedostoformaatti.
RAD	Rapid Application Development, ohjelmistojen kehittäminen tehokkaiden työkalujen avulla.
SQL	Structured Query Language, tietokannan kyselykieli.
UML	Unified Modeling Language, mallinnus- ja määrittelykieli.
WAR	Web Application Archive, pakettimainen tiedostoformaatti Java-pohjaisille web-sovelluksille.
WML	Wireless Markup Language, kannettavia päätelaitteita varten suunniteltu sivunkuvauskieli.
WWW	World Wide Web, Internetissä toimiva hypertekstijärjestelmä.
WYSIWYG	What You See Is What You Get, graafisten kehitysympäristöjen tarjoama visuaalinen näkymä, joka pyrkii esittämään suunniteltavan käyttöliittymän lopullisen ulkoasun.
XHTML	Extensible Hypertext Markup Language, HTML:stä kehitetty kuvauskieli, joka noudattaa XML:n muotovaatimuksia.
XML	Extensible Markup Language, yleiskäyttöinen W3C-konsortion hallinnoima kuvauskieli.

Johdanto

Nykyisin yhä useampien tietojärjestelmien käyttöliittymänä toimivat web-sivut. Yksi keskeinen merkitys web-sivujen suosioon on käyttöönoton ja käytön helppous, sivujen näyttämiseen tarvitaan vain web-selain eikä jokaista järjestelmää varten tarvitse rakentaa erillistä asiakasohjelmaa. Loppukäyttäjän kokema vaivattomuus ja lisäarvo eivät tule ilmaiseksi, sillä web-sovellusten kehittäminen on monimutkaista. Web-sovelluskehityksen vaikeudet johtuvat pitkälti siitä, että WWW (World Wide Web) alun perin suunniteltiin staattisten web-sivujen selailuun.

Web-sovelluskehityksessä eletään murroskautta, sillä käyttöliittymäkomponentit tuovat täysin uudenlaisen näkemyksen web-sivujen rakentamiseen. Web-sivu ei ole enää pelkkä ohjelmakoodia sisältävä määrittelydokumentti, vaan sivu voidaan luoda upottamalla siihen mitä erilaisimpia käyttöliittymäkomponentteja. Käyttöliittymäkomponentti voi olla esimerkiksi älykäs kalenteri-komponentti, joka web-sivulle lisättynä antaa loppukäyttäjälle mahdollisuuden selata kalenteria ja valita päivämäärän. Sovelluskehittäjän ei tarvitse edes tietää miten komponentti sisäisesti toimii, vaan hän voi komponentteja yhdistelemällä luoda hyvin monipuolisia web-käyttöliittymiä.

Käyttöliittymäkomponenttipohjaisia web-teknologioita on markkinoilla lukumääräisesti vähän, koska paradigma on vielä uusi ja isojen web-sovelluskehysten rakentaminen vaatii paljon aikaa ja resursseja. Komponenttiteknologiaa valittaessa on erityisen tärkeää huomioida teknologian yleisyys, tulevaisuuden näkymät, tarjolla olevat kehitystyökalut sekä teknologian taustavoimat. Juuri näiden syiden johdosta tähän työhön valittiin Microsoftin ASP.NET ja Java-yhteisön kehittämä JavaServer Faces. Näitä komponenttiteknologioita on markkinoitu mullistavina ja niiden pitäisi nopeuttaa ja helpottaa web-sovellusten kehittämistä merkittävästi. Sekä ASP.NET että JavaServer Faces mahdollistavat web-sivujen rakentamisen käyttöliittymäkomponenteilla. Teknologiat sisältävät joukon peruskomponentteja, mutta uusia komponentteja voi hankkia kolmansilta osapuolilta sekä rakentaa itse.

Työn tarkoituksena on selvittää mitä lisäarvoa käyttöliittymäkomponenttipohjainen kehitys tuo web-sovellusten kehittämiseen, ja pystyykö käyttöliittymäkomponentteja hyödyntämällä rakentamaan suorituskykyisen järjestelmän. Työn kokeellisessa osuudessa ASP.NET- ja JavaServer Faces- teknologioilla toteutetaan kaksi eri

prototyyppiä, jotka vastaavat erästä Nokialla tuotantokäytössä olevaa web-pohjaista raportointijärjestelmää. Prototyyppien avulla vertaillaan komponenttiteknologioiden suorituskykyä toisiinsa sekä vanhempaan web-teknologiaan pohjautuvaan toteutukseen. Toinen prototyypeistä valitaan jatkokehitykseen ja se tulee korvaamaan tuotannossa olevan raportointijärjestelmän.

Työn ensimmäisessä luvussa esitellään web-sovelluskehityksen historiaa, nykyisiä merkittäviä web-teknologioita sekä uudet komponenttiteknologiat ASP.NET ja JavaServer Faces.

Toisessa luvussa esitellään tuotantokäytössä oleva KP-järjestelmä, sen raportointiosuuden nykyinen toteutus ja tämän työn yhteydessä toteutetut kaksi prototyyppiä.

Kolmannessa luvussa suoritetaan komponenttiteknologioiden vertailu ja esitellään käytetyt testiympäristöt ja suorituskykymittausten tulokset.

Neljännessä luvussa selvitetään mitä hyötyä komponenttiteknologioiden käyttämisestä oli, ja mikä merkitys niillä on web-sovelluskehitykselle nyt ja tulevaisuudessa. Tämän lisäksi tehdään valinta, kumpi työn yhteydessä toteutetuista prototyypeistä otetaan jatkokehitykseen ja esitetään valintaan johtaneet syyt.

1 Web-sovelluskehitys

1.1 Johdanto

Tässä luvussa käydään läpi web-sovelluskehityksen historiaa ja esitellään eri teknologiasukupolvet. Luvun alkuosassa käsitellään ensimmäisen ja toisen sukupolven web-teknologioita ja niihin liittyviä ongelmia. Luvun loppupuolella tarkastellaan käyttöliittymäkomponentteja ja niihin liittyviä tapahtumia. Tämän jälkeen esitellään uudet käyttöliittymäkomponentteja hyödyntävät kolmannen sukupolven web-teknologiat ASP.NET ja JavaServer Faces.

1.2 Web-sovelluskehityksen lyhyt historia

Vuonna 1990 englantilainen Tim Berners-Lee kehitti hypertekstien siirtoprotokollan (eng. Hypertext Transfer Protocol, HTTP), josta tuli WWW:n perusta. HTTP on tilaton protokolla ja perustuu asiakas-palvelin -malliin [HTTP-1.1]. HTTP-protokollaa hyödyntävät web-palvelimet ja web-selaimet. Tyypillisesti web-palvelimen asiakasohjelma on web-selain. Asiakasohjelma ottaa yhteyden palvelimeen ja lähettää pyynnön, joka sisältää halutun resurssin osoitteen. Palvelin tulkitsee pyynnön ja palauttaa pyydetyn resurssin tai virhekoodin, tämän jälkeen yhteys palvelimeen katkaistaan. HTTP:n versiossa 1.1 asiakasohjelma voi pitää yhteyttä auki ja pyytää useamman resurssin peräkkäin, tätä hyödynnetään esimerkiksi tilanteessa, jossa web-sivu sisältää kuvia, jolloin web-selain pyytää myös kuvat avaamatta jokaista kuvaa varten uutta yhteyttä. Yhteyksien avaaminen ja sulkeminen on hidasta, joten tämä menettely nopeuttaa sivujen selailua. HTTP soveltuu hyvin staattisen tiedon jakamiseen. Hyvin pian WWW:n yleistyttyä haluttiin web-sivuilla kuitenkin näyttää myös dynaamista aineistoa, esimerkiksi elokuvateatterin sivuilla tuli julkaista päivän ohjelmisto. Ongelmaa ei haluttu ratkaista tuottamalla joka päivä uusi staattinen sivu, vaan web-palvelimen piti luoda web-sivu dynaamisesti sitä pyydettäessä. Tätä varten kehitettiin teknologioita, joilla HTTP-pyyntö voitiin ohjata web-palvelimella ajettavalle ohjelmalle, joka sitten tulosti vastauksen selaimelle. Aluksi suosituin tapa oli Common Gateway Interface (CGI).

Kun web-palvelimet pystyivät tarjoamaan dynaamista aineistoa, avautui uusia mahdollisuuksia. Staattisten sivujen sijasta tietoa voitiin hakea sivulle esimerkiksi

tietokannasta. Koska HTTP tukee GET-pyyntöjen lisäksi myös POST-pyyntöjä, tietoa voitiin lähettää myös selaimelta web-palvelimelle. Dynaamisen aineiston tuominen web-sivustoille asetti uusia haasteita sekä kehitystyölle että ylläpidolle - sivustot eivät olleet enää vain joukko staattisia tekstipohjaisia sivuja vaan osalla sivuista oli viittauksia palvelimilla ajettaviin ohjelmiin.

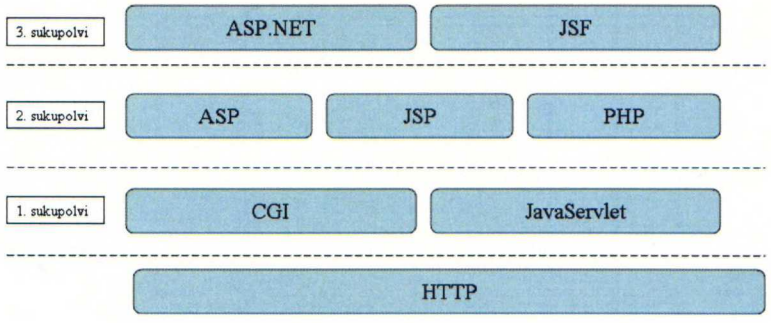
Vuonna 1996 Microsoft esitteli Active Server Pages -teknologian (ASP), joka toi uuden näkemyksen dynaamisten web-sivujen luontiin. ASP tarjosi mahdollisuuden lisätä web-sivulle logiikkaa skriptikielellä. Kun web-selain pyytää ASP-sivua, niin sivu esikäsitellään web-palvelimella ajamalla se ASP-moottorin lävitse. ASP-moottori suorittaa sivun sisältämät skriptit ja vasta sen jälkeen sivu lähetetään web-selaimelle. Skriptikielet ovat hyvin monipuolisia ja niillä voidaan suorittaa mm. tietokantahakuja ja lisätä dynaamista sisältöä web-sivulle. ASP toimitettiin Microsoftin Internet Information Server 3.0:n mukana eikä Microsoft tukenut ASP-teknologian käyttöä muiden valmistajien web-palvelimissa.

Sun Microsystems julkaisi vuonna 1999 JavaServer Pages -teknologian (JSP). JSP oli ASP:n tavoin tapa lisätä HTML-sivulle logiikkaa, kielenä käytettiin Javaa. JSP oli riippumaton käytetystä web-palvelimesta ja JSP-sivut käännettiin palvelmiksi (eng. servlets). Sekä JSP että ASP mahdollistivat istunnot eli tavan tunnistaa saman asiakkaan lähettämät HTTP-pyynnöt. Istunnot olivat suuri askel eteenpäin, koska niiden avulla web-sovellukset eivät olleet enää tilattomia, vaan dynaamiset sivut pystyivät reagoimaan käyttäjän aikaisempiin valintoihin. Istunnot toivat web-sovellukset askeleen lähemmäksi työpöytäsovelluksia, koska web-palvelimen muistiin voitiin tallentaa istuntoon liittyviä tietoja, esimerkiksi verkkokaupassa asiakkaan ostoskorin sisältö.

1.3 Perinteiset teknologiat

1.3.1 Yleistä

Perinteiset web-teknologiat voidaan jakaa karkeasti kahteen eri sukupolveen, jolloin uudempien komponenttiteknologioiden voidaan katsoa muodostavan kolmannen sukupolven kuten kuvassa 1 on esitetty. Ensimmäisen sukupolven teknologiat toimivat hyvin lähellä web-palvelinta, ne lukevat HTTP-pyyntöjä lähes raakamuodossa ja kirjoittavat vastauksen suoraan HTTP-yhteyden tietovirtaan. Toisen sukupolven web-teknologiat perustuvat kuvauskielisiin dokumentteihin (tyypillisesti HTML tai XHTML), jotka sisältävät palvelimessa ajettavia skriptejä. Dokumentti koostuu web-sivun ulkoasun määrittävistä kuvauskielisistä elementeistä, staattisesta tekstimuotoisesta sisällöstä ja dynaamisen sisällön luomiseen tarvittavista palvelinpään skripteistä. Kun web-palvelimelta pyydetään tällaista dokumenttia, niin sille suoritetaan esikäsittely, jolloin dokumentin sisältämät skriptit ajetaan. Skriptit pystyvät tulostamaan sisältöä HTTP-vastauksen tietovirtaan, joten vastauksen lopullinen muoto on yhdistelmä staattista sisältöä ja skriptien tuottamaa sisältöä. Perinteisille web-teknologioille on tyypillistä, että liiketoimintalogiikka sijoitetaan samaan paikkaan kuin web-sivujen esityslogiikka, joka vaikeuttaa sivujen ylläpitoa. Suurta huolellisuutta noudattaen liiketoimintalogiikka voidaan erottaa esityskerroksesta, mutta se on hankalaa verrattuna uusiin kolmannen sukupolven komponenttiteknologioihin.



Kuva 1: Tunnetuimmat web-teknologiat sukupolvittain.

1.3.2 Common Gateway Interface

Common Gateway Interface (CGI) on alun perin NCSA:n kehittämä teknologia ja se julkaistiin vuonna 1993. CGI-standardi määrittelee rajapinnan miten web-palvelin

voi käynnistää ulkoisen ohjelman palvelemaan HTTP-pyyntöä [CGI-W]. Ulkoinen ohjelma voi olla käännetyssä muodossa (esimerkiksi C-ohjelma) tai tulkattava skripti (esimerkiksi Perl-ohjelma). Ohjelma ajetaan itsenäisenä prosessina, eikä se pysty vuorovaikuttamaan web-palvelimen kanssa. Ohjelma pystyy lukemaan tiettyjä HTTP-pyynnön otsikkokenttiä ympäristömuuttujien kautta ja kirjoittamaan vastauksen suoraan standarditulostusvirtaan, jonka web-palvelin ohjaa HTTP-vastausvirtaan. Jokaista HTTP-pyyntöä varten käynnistetään oma prosessi, joten CGI-ohjelmat kuluttavat paljon palvelimen resursseja eivätkä näin skaalaudu hyvin. CGI oli yksi ensimmäisistä tavoista tuoda web-sivuilla dynaamista sisältöä ja CGI saavuttikin de facto -standardin aseman etenkin Unix-ympäristöissä. CGI:n hyvä ominaisuus oli riippumattomuus ohjelmointikielestä, sillä CGI-ohjelman pystyi kirjoittamaan lähes millä tahansa ohjelmointikielellä. CGI:n huonoja puolia olivat skaalautumattomuus ja tietoturva. Huonosti toteutetut CGI-ohjelmat saattoivat vaarantaa koko palvelimen tietoturvan. CGI:tä hyödyntävien web-sovellusten ylläpito oli hankalaa, esimerkiksi jos tulosteen muotoa haluttiin muuttaa, täytyi muutos tehdä ensin CGI-ohjelman lähdekoodiin ja sitten kääntää se uudelleen. CGI:stä kehitettiin vaihtoehtoisia toteutuksia kuten FastCGI [FastCGI-W], joiden tarkoituksena oli parantaa CGI:n skaalautuvuutta. Useat valmistajat liittivät web-palvelinohjelmistoihin CGI-laajennuksia, joita varten tehdyt CGI-ohjelmat toimivat vain valmistajien omissa ympäristöissä.

1.3.3 Active Server Pages

Active Server Pages (ASP) on Microsoftin vuonna 1996 julkaisema toisen sukupolven web-teknologia. ASP toimii Microsoftin Internet Information Server web-palvelinohjelmiston (IIS) laajennuksena. ASP perustuu HTML-sivuihin, joihin voidaan sisällyttää palvelimella ajettavia skriptejä. Microsoftin tukemia skriptikieliä ovat VBScript ja JScript. Kolmansilta osapuolilta voi hankkia tuen muille kielille kuten Perlille. Ylivoimaisesti suosituin ASP-sivujen skriptikielistä on VBScript, joka on Visual Basic -kielen osajoukko. VBScript tukee olioita, mutta sitä ei voi pitää olio-ohjelmointikielenä, koska siitä puuttuu olennaisia olio-ominaisuuksia kuten perintä ja rajapinnat. VBScript sisältää suppean funktiokirjaston tavallisimpiin toimintoihin kuten merkkijonojen käsittelyyn ja matemaattisiin operaatioihin. VBScriptillä pääsee käsiksi myös ASP-olioihin, joilla pystyy mm. lukemaan HTTP-

pyyntöjen parametreja, hallinnoimaan istuntoja ja varastoimaan palvelimen muistiin kaikille käyttäjille yhteisiä tietoja.

ASP:tä ei voida pitää tapahtumaperusteisena vaikkakin se tarjoaa muutaman tapahtumankäsittelijän, joilla istunnon ja itse sovelluksen käynnistyminen ja loppuminen voidaan havaita. ASP:tä ei voida pitää myöskään komponenttipohjaisena, vaikka se sisältää joitakin palvelinpään komponentteja mm. mainosten näyttämiseen ja kävijämäärän laskemiseen. ASP:n vahvuutena on hyvä yhteistoiminta Microsoftin ActiveX- ja COM-komponenttiteknologioiden kanssa. ASP-sovellus voi vaivatta keskustella esimerkiksi tietokantojen ja viestijonojen kanssa. ASP:n heikkouksia ovat skriptien tulkkaus (hidasta), rajoittuneet olio-ominaisuudet ja alustariippuvuus. Microsoft ei enää kehitä ASP:tä vaan ASP.NET tulee korvaamaan sen täysin, näin viimeiseksi versioksi jää 3.0.

1.3.4 JavaServlets ja JavaServer Pages

Vuonna 1997 Sun Microsystems sai valmiiksi ensimmäisen JavaServlet-standardin ja kaksi vuotta myöhemmin julkaistiin JavaServer Pages -teknologia. JavaServlet-teknologia mahdollistaa CGI-tyyppisten ohjelmien tekemisen Javalla. Ohjelmat koostuvat Java-servleteistä, jotka vaativat Java-yhteensopivan web-palvelimen tai itsenäisen JavaServlet-säiliön (eng. stand alone JavaServlet container). Web-palvelin keskustelee Java-servletin kanssa Servlet-ohjelmointirajapinnan kautta ja välittää vastaanotetut HTTP-pyynnöt servletille ja palauttaa servletin antamat vastaukset asiakkaalle. Servlet-teknologiaa ei ole sidottu pelkästään HTTP:hen vaan se soveltuu myös muiden palvelinohjelmien toteuttamiseen, joita voi olla esimerkiksi FTP-palvelinohjelma.

Java-servletit päihittivät suorituskyvyssä CGI-ohjelmat, koska servlettejä ajetaan web-palvelimen prosessiavaruudessa eikä jokaista HTTP-pyyntöä varten käynnistetä uutta prosessia. Java-servletit tukevat istuntoja ja niillä on käytettävissä Javan kattavan luokkakirjaston lisäksi servlet-kirjasto, josta löytyy valmista toiminnallisuutta mm. HTTP-pyyntöjen käsittelyyn. Java-servletit ovat alustariippumattomia ja niitä voidaan ajaa missä tahansa Java-yhteensopivassa web-palvelimessa. Servlettien tietoturvaa pidetään hyvänä, koska ohjelmien turvallisuudesta ja oikeuksista vastaa Java-virtuaalikone.

JavaServlet-teknologia sopi hyvin CGI-ohjelmien korvaajaksi, mutta web-sivustojen toteuttaminen servleteillä oli hankalaa, koska HTML-elementit ja sisältö täytyi

tuottaa ohjelmallisesti. Sun Microsystems ratkaisi ongelman kehittämällä JavaServer Pages -teknologian (JSP). JSP on Microsoftin ASP:n kaltainen teknologia, jolla web-sivut toteutetaan tekstipohjaisella kuvauskielellä ja dynaaminen aineisto luodaan sivujen lähdekoodiin upotetulla ohjelmointikielellä. JSP käyttää ohjelmointikielenä Javaa, mutta logiikkaa voi lisäksi toteuttaa myös erityisillä kuvauskielisillä komennoilla (JSP actions ja JSP Tag Libraries). JSP-sivuja ei tulkata ASP:n tapaan vaan ne käännetään erityisellä kääntäjällä servleteiksi, täten JSP on servlettejä täydentävä teknologia. Monet avoimen lähdekoodin yhteisöt ovat laajentaneet JavaServlet- ja JSP-teknologioita ja luoneet niihin pohjautuvia web-sovelluskehyksiä (esimerkiksi Struts). Kyseiset sovelluskehykset helpottavat web-sovellusten rakentamista, mutta eivät ole käyttöliittymäkomponenttipohjaisia.

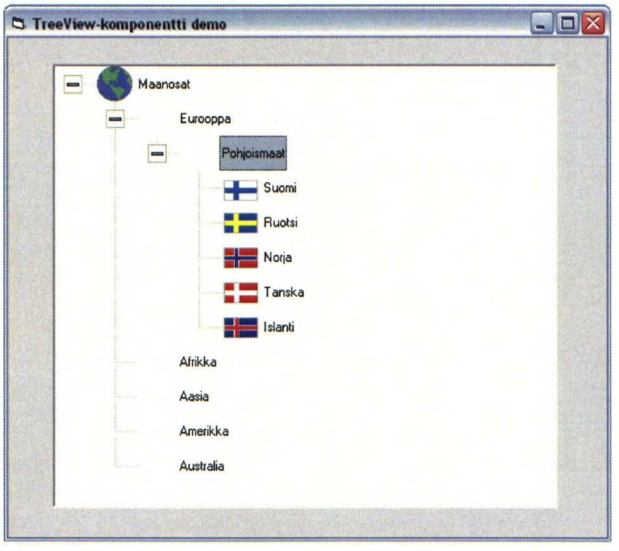
1.3.5 PHP: Hypertext Preprocessor

PHP on avoimeen lähdekoodiin perustuva toisen sukupolven web-teknologia. Teknisesti PHP on sukua ASP:lle ja JSP:lle. Web-sivut toteutetaan kuvauskielisinä dokumentteina (esimerkiksi HTML tai XHTML) ja dynaaminen sisältö tuotetaan dokumenttiin upotetuilla PHP-kielisillä skripteillä. PHP-kieli on dynaamisesti tyypitetty ja syntaksiltaan lähellä Perliä ja C:tä. PHP:tä pidetään helposti omaksuttavana ja se voi olla yksi selitys sen laajaan suosioon, toinen mahdollinen selitys on ilmaisuus ja tuki usealle eri alustalle. Alkujaan PHP:ssä oli heikko olio-tuki, mutta versiossa 5.0 olio-ominaisuuksia on huomattavasti parannettu. PHP tarjoaa varsin kattavan tuen yleisimmille tietokannoille ja sille löytyy funktio-kirjastoja lähes mihin tahansa ohjelmointitarkoitukseen.

1.4 Nykyinen ongelmakenttä web-sovelluskehityksessä

Web-sovelluksilta vaaditaan nykyisin yhä enemmän, niiden käyttöliittymien täytyy olla monipuolisia ja helppokäyttöisiä. Usein vertailukohtana käytetään työpöytäsovelluksia, joiden arkkitehtuuri on kuitenkin aivan erilainen. Työpöytäsovellusten rakentamiseen löytyy graafisia kehitysympäristöjä, joilla käyttöliittymiä voidaan luoda nopeasti ja tehokkaasti (ns. RAD-kehitys) [RAD-W].

Työpöytäsovellusten käyttöliittymäteknologiat kuten Java Swing, Microsoft Visual Basic Forms ja Borland Delphi ovat komponenttipohjaisia ja tapahtumaperusteisia. Komponenttipohjaisuus tarkoittaa sitä, että käyttöliittymä voidaan rakentaa itsenäisiä komponentteja yhdistelemällä. Komponentti on looginen kokonaisuus, jolla on tietty toiminnallisuus, sisäinen tietomalli ja ulkoinen rajapinta. Käyttöliittymien yhteydessä puhutaan käyttöliittymäkomponenteista, joilla on yleensä myös visuaalinen ulkoasu. Kuvassa 2 on esitetty esimerkki ikkunaan upotetusta Visual Basicin TreeView-nimisestä käyttöliittymäkomponentista, jolla voidaan esittää hierarkkista tietoa puurakenteena, komponenttiin voi sisällyttää myös kuvia.



Kuva 2: TreeView-komponentilla tietoa voidaan esittää puurakenteena.

Tapahtumaperusteisuus tarkoittaa sitä, että käyttäjän tekemät valinnat käyttöliittymässä aiheuttavat tapahtumia. Tapahtuma voi olla esimerkiksi napin painallus tai alkion valitseminen listasta. Tapahtumat välittyvät tapahtumankäsittelijälle, jossa päätetään miten tapahtumiin reagoidaan.

Perinteiset web-teknologiat eivät ymmärrä tapahtumia vaan ne hyödyntävät vain HTTP-pyyntöjä ja -vastauksia, esimerkiksi kun käyttäjä syöttää web-sivun lomakkeeseen tiedot ja painaa lähetä-nappia, niin web-sovellus vastaanottaa HTTP POST -pyynnön. Pyyntö sisältää lomakkeen kentät ja niiden arvot, jotka web-sovelluksen pitää parsia. Tapahtumapohjainen sovellus tietäisi, että käyttäjä painoi lähetä-nappia ja samalla se voisi lukea käyttäjän täyttämät arvot tai reagoida jotenkin muuten tapahtumaan. Perinteisissä web-teknologioissa web-sivujen rakenne muodostetaan HTML-kielellä tai vastaavalla kuvauskielellä. Napit, valikot ja listat ovat sovelluksen kannalta vain kuvauskielellä määriteltyjä elementtejä. Perinteisten web-teknologioiden avulla monipuolisten web-käyttöliittymien tekeminen on hidasta ja ylläpito vaikeaa verrattuna työpöytäohjelmien luomiseen tehokkailla kehitysympäristöillä. Suurin ongelman aiheuttaja on se, että perinteisissä web-teknologioissa sivujen esityskerros on sidoksissa tiukasti logiikkakerrokseen, käytännössä tämä ilmenee määrittelydokumenttina, joka sisältää sekaisin ulkoasun muotoilukomentoja sekä dynaamisen sisällön luomiseen tarvittavia skriptejä.

1.5 Uudet teknologiat

1.5.1 Yleistä

Vuonna 2002 Microsoft julkaisi ASP.NET-teknologian osana .NET-sovelluskehystä. ASP.NET mahdollistaa komponenttipohjaiset, tapahtumaperusteiset web-sovellukset ja sen tarkoitus on yksinkertaistaa web-sovelluskehitys lähes samalle tasolle, jolla työpöytäsovelluskehitys on nykyisin. ASP.NET toimii vain Microsoftin ympäristöissä vaikkakin on olemassa joitakin avoimeen lähdekoodiin perustuvia projekteja, joiden tarkoituksena on ulottaa .NET ja samalla ASP.NET muillekin alustoille [Mono-W] [DotGNU-W]. Vuonna 2004 valmistui Java-yhteisön (eng. Java Community Process) toimesta JavaServer Faces (JSF) -spesifikaatio, joka määrittelee uuden tavan tehdä web-sovelluksia Java-ympäristöihin. Tärkeimmät uudet ominaisuudet ovat komponenttipohjaisuus ja tapahtumaperusteisuus. Molempien uusien teknologioiden avulla pyritään merkittävästi helpottamaan web-käyttöliittymien rakentamista. Web-sivun visuaalinen malli on erotettu sen taakse kytketystä logiikasta käyttämällä sivua avustavia luokkia, JSF:n tapauksessa käytetään nimitystä "managed bean" tai "backing bean" ja ASP.NETissä käsitettä "code-behind". Kehittäjille tarjotaan joukko valmiita käyttöliittymäkomponentteja, joita yhdistelemällä käyttöliittymän rakentamisen tulisi olla nopeaa ja vaivatonta.

1.5.2 Käyttöliittymäkomponentit web-sovelluksissa

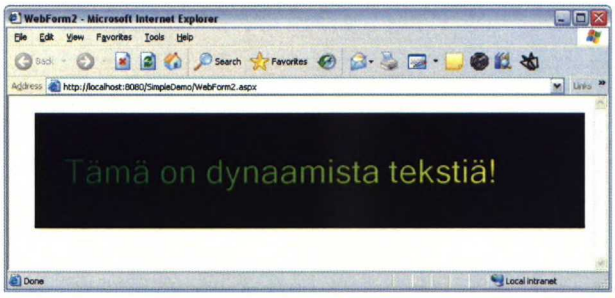
Käyttöliittymäkomponentteja on jo pitkään hyödynnetty työpöytäsovelluksissa ja graafisten käyttöjärjestelmien ikkunointijärjestelmien toteutuksissa. Niiden tehtävänä on kommunikoida käyttäjän kanssa, esittää ja vastaanottaa tietoa. Käyttöliittymäkomponentit reagoivat käyttäjän tekemiin valintoihin synnyttämällä tapahtumia, jotka sovellus sitten käsittelee tai jättää käsittelemättä. Kehittäjän näkökulmasta käyttöliittymäkomponentti kapseloi tietyn toiminnallisuuden ja tarjoaa ulkoisen rajapinnan komponentin tilan tai ominaisuuksien muokkaamiseen ohjelmallisesti. Käyttöliittymäkomponentilla on yleensä visuaalinen ulkoasu, johon voi vaikuttaa sen julkisten ominaisuuksien ja metodien kautta, komponentti osaa myös usein piirtää itsensä. Komponentin ominaisuus kertoo komponentin tilasta, esimerkiksi tekstikentän ominaisuuksia voisi olla tekstin väri ja koko. Joitakin ominaisuuksia ei välttämättä pysty muuttamaan komponentin ulkopuolelta, joten ne ovat vain luettavissa. Metodit ovat komponentin tarjoamia toimintoja, joita kutsumalla

komponentin voi pyytää suorittamaan jonkin toimenpiteen, esimerkiksi älykäs tekstinsyöttökomponentti voisi tarjota metodin, jota kutsumalla se tarkistaa sisältääkö annettu tekstisyöte kielioppivirheitä. Monimutkaisemmat komponentit pystyvät julkaisemaan tapahtumia ja rekisteröimään tapahtumien kuuntelijoita. Tapahtuma aiheutuu yleensä kun käyttäjä tekee jotain käyttöliittymäkomponentille, esimerkiksi valitsee listasta arvon. Tapahtuma havaitaan web-sovelluksessa ja siihen mahdollisesti liitetty ohjelmalogiikka suoritetaan. Käyttöliittymäkomponenteilla käyttöliittymien rakentaminen on nopeaa, erityisesti graafisissa kehitysympäristöissä komponentteja voidaan asetella vedä-ja-pudota -periaatteella suoraan näkymään.

Web-sovelluksen asiakasohjelmana on tyypillisesti web-selain, joka on erikoistunut HTML:n tai muiden kuvauskielten näyttämiseen. Jotta käyttöliittymäkomponentti voidaan näyttää web-selaimessa, on komponentti pystyttävä esittämään jollain selaimen ymmärtämällä kuvauskielellä. Tästä johtuen web-sovelluksissa käytetyt käyttöliittymäkomponentit eroavat merkittävästi työpöytäsovelluksissa käytetyistä komponenteista, joiden piirtäminen tapahtuu käyttöjärjestelmän grafiikkakirjastojen avulla. Käyttöliittymäkomponentista on ajonaikainen malli web-sovelluksen muistissa, tällaisesta mallista käytetään usein nimitystä oliomalli. Muuntamista ajonaikaisesta oliomallista kuvauskielelle kutsutaan renderöinniksi. Renderöinnin tuloksena komponenttien ominaisuudet pyritään ilmaisemaan kuvauskielellä, esimerkiksi komponentin taustaväri voi olla oliomallissa RGB-muodossa, mutta kuvauskielisessä esityksessä väri voidaan ilmaistaan nimeltä tekstimuodossa. Komponentti voi renderöidä itsensä, mutta muunnos voidaan tehdä myös komponentin ulkopuolella. Muunnos kuvauskieliseen esitykseen ei välttämättä ole suoraviivainen ja ennaltamäärätty, vaan se voidaan tehdä kohdelaitteen ominaisuuksien perusteella. Komponentti voi esimerkiksi renderöidä itsensä HTML-kielellä web-selaimelle, mutta WML-kielellä mobiililaitteelle. Muunnoksen ei täydy tapahtua pelkästään tietylle tekstipohjaiselle kuvauskielelle vaan käännös esimerkiksi vektorigrafiikaksi voi olla mahdollista.

Sekä ASP.NET ja JavaServer Faces sisältävät peruskomponentit, joita tyypillisessä web-sovelluksessa tarvitaan. Tällaisia komponentteja ovat mm. napit, hyperlinkki-napit, valintalaatikot, tekstilaatikot ja valintalistat. Komponenttiteknologian yksi hyödyllisimmistä puolista on kuitenkin se, että kehittäjät pystyvät luomaan uusia komponentteja tai käyttämään kolmansien osapuolten rakentamia komponentteja. Työpöytäsovelluksille komponenttitarjonta on hyvin runsasta ja luonut oman

teollisuudenalan. Kuvassa 3 on esimerkki kolmannen osapuolen valmistamasta ASP.NET-käyttöliittymäkomponentista, johon voidaan piirtää dynaamisesti grafiikkaa ja tekstiä grafiikkakirjaston piirtofunktioilla, komponentti renderöi kuitenkin itsensä selaimelle staattiseksi kuvaksi JPEG-, BMP-, tai PNG-formaattiin.



Kuva 3: Silk Webwaren Image Canvas -niminen ASP.NET-käyttöliittymäkomponentti osaa esittää dynaamisesti luotua grafiikkaa staattisina kuvina.

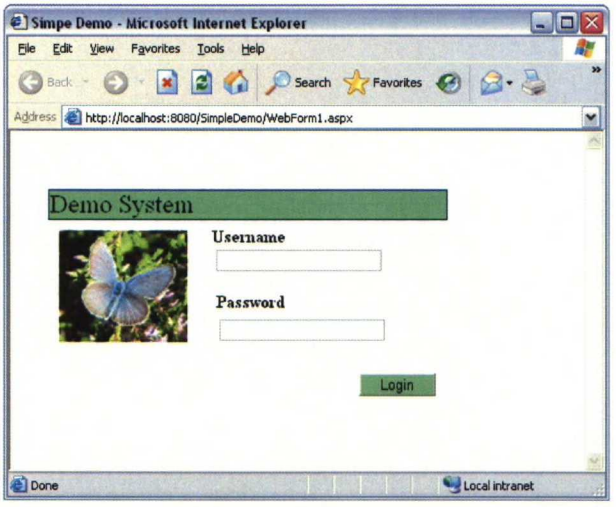
Uusia käyttöliittymäkomponentteja voidaan luoda pienellä vaivalla joko laajentamalla peruskomponentteja tai yhdistelemällä olemassa olevista komponenteista suurempia kokonaisuuksia. Komponentteja voidaan myös rakentaa puhtaalta pöydältä, mutta se on yleensä työläämpää.

1.5.3 Komponenttipuut

Web-sivulle sisällytetyt käyttöliittymäkomponentit muodostavat hierarkian, jota kutsutaan komponenttipuuksi. Jotkin komponentit toimivat vain säiliöinä muille komponenteille. Kullakin komponentilla on yksilöivä tunniste, jolla komponentit voidaan erottaa toisistaan. Kun käyttöliittymäkomponentteja sisältävä web-sivu pyydetään web-palvelimelta, niin web-sovelluksessa muodostetaan komponenttipuun ajonaikainen malli web-sivun kuvauskielisen määrityksen pohjalta. Ajonaikainen malli koostuu olioista, joilla on isä-komponentti ja kokoelma lapsikomponentteja. Oliot ovat ilmentymiä komponenttikirjaston luokista. Komponenttipuun juurena on yleensä web-sivua esittävä isätön olio. Kun web-sivu lähetetään asiakasohjelmalle kuten web-selaimelle, niin web-sovelluksen renderöintivaiheessa koko komponenttipuu käydään lävitse ja kukin komponentti tuottaa kuvauskielisen esityksen itsestään ja sen hetkisestä tilastaan.

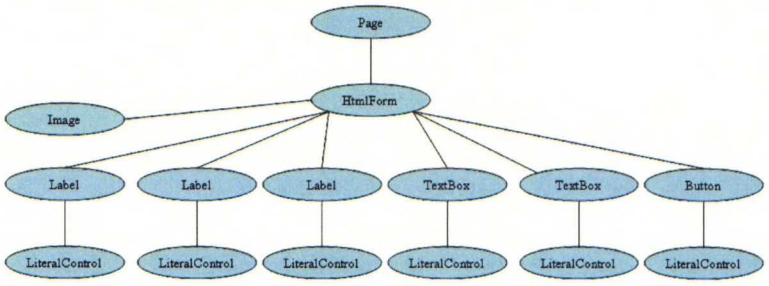
Komponenttipuu mahdollistaa käyttöliittymäkomponenttien muokkaamisen ohjelmakoodilla. Puussa olevien komponenttien ominaisuuksia voidaan lukea ja muuntaa, puuhun voidaan myös dynaamisesti lisätä uusia komponentteja tai poistaa siellä jo

olevia komponentteja. Osia komponenttipuusta voidaan jättää piirtämättä asiakasohjelmalle, tämä mahdollistaa yksinkertaisemman ja käyttäjän valintoihin mukautuvan käyttöliittymän rakentamisen. Kuvassa 4 on esitetty esimerkki yksinkertaisesta web-sivusta, joka voisi toimia jonkin tietojärjestelmän kirjautumissivuna. Sivulla on kolme tekstikenttää, kaksi tekstilaatikkoa, kuva ja painonappi.



Kuva 4: Esimerkki web-sivusta.

Kuvassa 5 on esitetty saman web-sivun komponenttipuu, jos sivu olisi toteutettu ASP.NET-sivuna. Komponenttipuun juurena on sivua esittävä komponentti, joka sisältää joukon tekstin näyttämiseen ja syöttämiseen tarkoitettuja komponentteja, painonappi-komponentin ja kuvan näyttämiseen tarkoitetun komponentin. Label-, TextBox- ja Button-komponentit sisältävät vielä yhden lapsikomponentin, joka edustaa komponentin tekstiarvoa.



Kuva 5: Esimerkki komponenttipuusta.

1.5.4 Tapahtumien käsittely

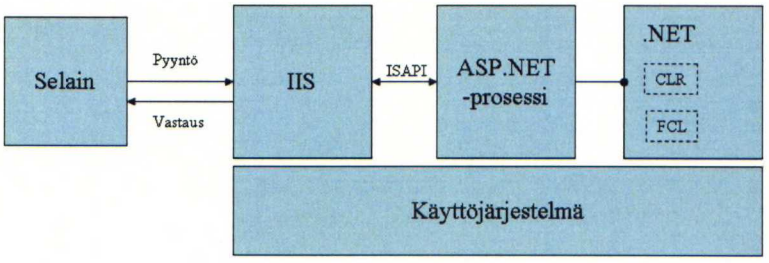
Kun käyttäjä painaa hiiren painiketta painonappi-komponentin päällä web-sivulla, niin se on esimerkki tapahtumasta. Painonappiin voidaan liittää useita eri tapahtumia, esimerkiksi hiiren tuominen napin päälle voi myös olla tapahtuma. Tapahtumasta aktivoituvaa ohjelmakoodia kutsutaan tapahtumankäsittelijäksi. Tapahtumat ovat keino ohjata ohjelman suoritusta ja reagoida käyttäjän tekemiin valintoihin. Tapahtumapohjaisen ohjelmoinnin katsotaan olevan ohjelmointiparadigma, jolla ohjelmista saadaan tilattomampia. Tapahtumien käyttäminen helpottaa erityisesti graafisen käyttöliittymän rakentamista, jolloin tapahtumia tuottavat käyttöliittymä-komponentit. Tilattomuus tarkoittaa tässä yhteydessä, että sovellus voi saada komentoja käyttöliittymän kautta lähes missä tahansa järjestyksessä. Useimmat työpöytäsovellusten graafisten käyttöliittymien rakentamiseen soveltuvat teknologiat kuten Visual Basic, Delphi, Javan Swing-kirjasto ja .NET Winforms tukevat tapahtumia. Sekä ASP.NET että JavaServer Faces sisältävät kattavan tuen tapahtumille.

Olio-ohjelmoinnissa tapahtumien käsittelyyn hyödynnetään usein Tarkkailija-suunnittelumallia (eng. Observer pattern) [GHJV01]. Malli mahdollistaa olion toimimisen subjektina ja siitä kiinnostuneiden olioiden toimimisen tarkkailijoina. Kun subjektina olevan olion tila muuttuu, niin se ilmoittaa tarkkailijoille muutoksesta, tarkkailijat voivat sitten päivittää tietojaan oliosta. Tarkkailijat rekisteröivät itsensä subjektille ja pystyvät myös lopettamaan tarkkailun perumalla rekisteröinnin. Toisin kuin työpöytäsovelluksissa, käyttäjien aiheuttamia tapahtumia ei web-sovelluksissa kannata aina välittömästi raportoida palvelimelle, vaan muutokset voidaan välittää vasta seuraavassa selaimen suorittamassa HTTP POST -lähetyksessä.

1.5.5 ASP.NET

1.5.5.1 Yleistä

ASP.NET on Microsoftin kehittämä kolmannen sukupolven web-teknologia ja osa Microsoftin .NET-tuoteperhettä. ASP.NETillä voidaan rakentaa dynaamisia web-sivustoja hyödyntäen käyttöliittymäkomponentteja ja tapahtumia. Microsoft käyttää ASP.NET-sivuista nimitystä Web Forms. ASP.NETin arkkitehtuuri on kuvattu karkealla tasolla kuvassa 6. ASP.NET-ohjelmat toimivat Microsoftin Internet Information Server web-palvelimen (IIS) avustuksella. IIS välittää selaimelta tulevan ASP.NET-sivupyynnön ASP.NET-prosessille, joka käsittelee pyynnön ja tuottaa vastauksen. ASP.NET-pyynnöt tunnistetaan oletuksena aspx-päätteestä, myös muiden sivupäätteiden rekisteröinti on mahdollista. IIS ja ASP.NET-prosessi kommunikoivat ISAPI-nimisen rajapinnan kautta. ASP.NET-ohjelmakoodin suorittamisesta vastaa .NETin ajonaikainen ympäristö (eng. .NET Common Language Runtime) ja ohjelmat käyttävät .NET-luokkakirjastoa (eng. .NET Framework Class Library).



Kuva 6: ASP.NETin arkkitehtuuri.

ASP.NET-ohjelmia voidaan kirjoittaa useilla eri .NET-kielillä ja ne käyttävät .NET-luokkakirjaston tarjoamia palveluita. ASP.NET-sivu toteutetaan tyypillisesti kahtena eri tiedostona, toisessa tiedostossa on sivun kuvauskielinen määritys ja toisessa niin kutsutussa taustatiedostossa (eng. Code Behind file) on sivun ohjelmakoodi. Tarkoituksena on erottaa web-sivun ulkoasun esitys sen käyttöliittymälogiikasta, tämä on iso parannus verrattuna ASP-sivuihin, joissa ohjelmakoodi upotettiin kuvauskielen sekaan. ASP.NET tukee ohjelmakoodin upottamista myös kuvaustiedostoon, jotta vanhojen ASP-sovellusten päivittäminen sujuisi helpommin. ASP.NET-prosessi muodostaa sivumäärityksestä .NET-kielisen lähdekoodin, jossa käyttöliittymäkomponenteista luodaan ilmentymät ja komponenttipuun määrittävä

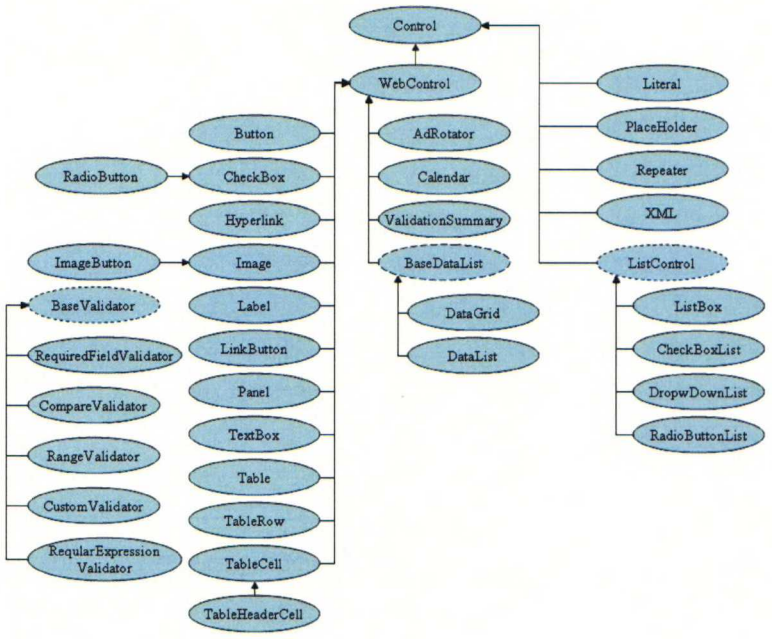
hierarkia. Lähdekoodi käännetään yhdessä taustatiedoston kanssa .NET-välikieliseksi DLL-tiedostoksi [Gur04-W].

Suosituimmat .NET-kielet ASP.NET-sivujen logiikan ohjelmointiin ovat C# ja Visual Basic.NET. Molemmat kielet ovat olio-ohjelmointikieliä ja käyttävät yhteistä .NET-luokkakirjastoa. ASP.NET-sivut kuten muutkin .NET-ohjelmat käännetään välikielelle, joka mahdollistaa kielten välisen yhteistoiminnan. Microsoftin tarjoamia muita .NET-kieliä ovat J#, JScript.NET ja Managed C++. Kolmansien osapuolten .NET-kielitarjonta on runsasta, mutta kaikki kielet eivät välttämättä tue ASP.NET-ohjelmien kehittämistä.

1.5.5.2 ASP.NET-käyttöliittymäkomponentit

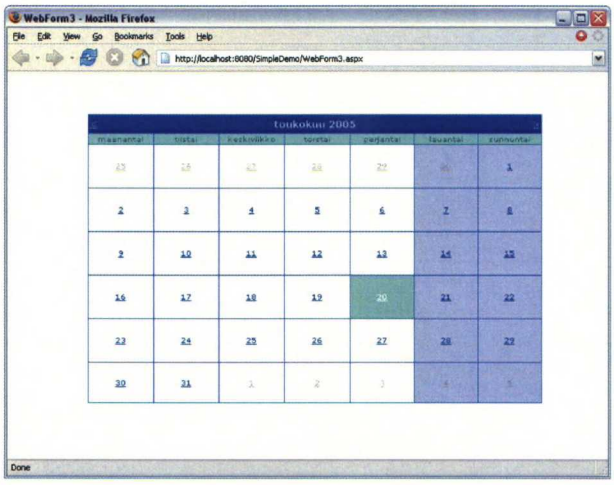
ASP.NET-sivut voidaan rakentaa kokonaan käyttöliittymäkomponenteilla tai niin, että käyttöliittymäkomponentteja sisällytetään tavalliseen HTML-sivuun vain tarvittavin paikkoihin. ASP.NET tukee perinteisiä HTML-käyttöliittymä-komponentteja ja niitä voidaan ajaa myös palvelimella, jolloin niihin on mahdollista viitata web-sovelluksen ohjelmakoodissa. Komponentit ovat kuitenkin hyvin pelkistettyjä, eikä niiltä löydy ominaisuuksia, joilla voisi määrittää esimerkiksi komponentin taustavärin. HTML-käyttöliittymäkomponentit helpottavat vanhojen sovellusten päivittämistä ASP.NET-alustalle, sillä ne vastaavat HTML-standardista löytyviä lomakkeille sijoitettavia elementtejä, esimerkiksi HtmlInputButton-niminen komponentti merkitään kuvaustiedostoon input-elementtinä, jonka type-attribuutiin arvona on "button".

Varsinaiset ASP.NET-käyttöliittymäkomponentit voidaan jakaa kolmeen eri tyyppiin [LH03]. Ensimmäisen tyypin komponentteja ovat Microsoftin tuottamat ASP.NETin peruskomponentit, joita on hieman yli 30 kappaletta. Toisen tyypin komponentteja ovat käyttäjäkomponentit (eng. user controls) ja kolmannen tyypin komponentteja kutsutaan räätälöidyiksi komponenteiksi (eng. custom controls). Peruskomponenttien hierarkia on esitetty kuvassa 7.



Kuva 7: ASP.NET-peruskomponenttien hierarkia.

Peruskomponentit sisältävät yleisimmät HTML-lomakkeissa käytetyt komponentit kuten tekstikentät, tekstialueet, painonapit, listat ja alasvetovalikot. Näiden HTML-standardissa [HTML-4.01] määriteltyjen komponenttien lisäksi löytyy komponentteja, jotka on tarkoitettu astetta monimutkaisempiin tarkoituksiin, eikä niitä voida renderöidä yhdeksi HTML-elementiksi. Esimerkkinä kuvassa 8 on esitetty Kalenteri-komponentti (eng. Calendar control), joka näyttää kuukauden päivät. Komponentti tarjoaa mahdollisuuden selata kuukausia ja valita päivämäärän. Komponentin ulkoasu on muokattavissa sen ominaisuuksien kautta, esimerkiksi värityksen voi valita ennalta määritellyistä värikokonaisuuksista tai säätää osa kerrallaan. Komponentti renderöi itsensä selaimelle tavalliseksi HTML-taulukoksi.



Kuva 8: Kalenteri-käyttöliitymäkomponentti.

Peruskomponentteihin kuuluu myös joukko käyttäjän antaman syötteen validointiin tarkoitettuja komponentteja. Validoinnit voivat olla yksinkertaisia tarkistuksia, että syöte on annettu tai syötteen numeerinen arvo on rajojen sisällä. Monimutkaisemmissa validaatioissa syötteen muoto voidaan tarkistaa säännöllisillä lausekkeilla. Validaatio-komponentille kerrotaan minkä käyttöliittymäkomponentin arvon sen pitää tarkistaa ja komponentti näyttää virheviestin mikäli arvo ei ole hyväksyttävä. Validaatio-komponentit osaavat tuottaa selaimessa ajettavaa skriptikieltä, jolloin validaatio suoritetaan asiakkaan päässä ennen kuin selain lähettää tiedot web-palvelimelle. Selaimen suorittamasta tarkistuksesta huolimatta syöte tarkastetaan vielä toisen kerran palvelimella, jottei tarkistusta pääse kiertämään.

Käyttäjäkomponentit ovat uudelleenkäytettäviä ASP.NET-sivun osia, joita voidaan sisällyttää varsinaisille ASP.NET-sivuille. Käyttäjäkomponentit rakennetaan yhdistelemällä olemassa olevia käyttöliittymäkomponentteja ja HTML-elementtejä. Käyttäjäkomponentit tukevat julkisia ominaisuuksia ja metodeita, ne pystyvät myös julkaisemaan tapahtumia. Samalle ASP.NET-sivulle voi sisällyttää eri .NET-kielillä tehtyjä käyttäjäkomponentteja.

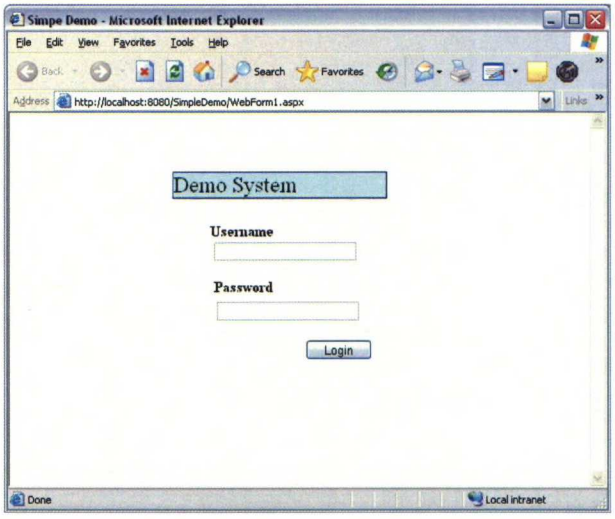
Räätälöidyt komponentit tarjoavat käyttäjäkomponentteja paremman tavan rakentaa uudelleenkäytettäviä käyttöliittymäkomponentteja, koska räätälöidyt komponentit voidaan kääntää itsenäisiksi ja helposti levitettäviksi DLL-tiedostoiksi. Räätälöidyt komponentit toimivat samaan tapaan kuin ASP.NETin peruskomponentit ja useat komponenttivalmistajat tarjoavatkin maksullisia komponentteja mitä erilaisimpiin tarkoituksiin. Räätälöityjä komponentteja voi hankkia kolmansilta osapuolilta tai

rakentaa itse. Räätälöity komponentti voidaan rakentaa kolmella eri tavalla. Ensimmäinen tapa on periyttää jokin ASP.NETin peruskomponentti ja laajentaa sen toiminnallisuutta, esimerkiksi tekstikenttä voitaisiin muuttaa vain numeroita hyväksyväksi. Toinen tapa on luoda komposiittikomponentti, joka voi olla yhdistelmä ASP.NETin peruskomponentteja ja räätälöityjä komponentteja. Kolmas tapa on työläin, mutta tarjoaa eniten joustavuutta ja toteutuksen vapautta. Siinä räätälöity komponentti luodaan periyttämällä WebControl-niminen juuri-komponentti, tällöin komponentin renderöinti HTML-kielelle on määriteltävä eksplisiittisesti. Jos komponentin halutaan säilyttävän tilansa HTTP-pyyntöjen välillä, on toteuttajan huolehdittava itse tilanhallinnasta.

ASP.NET-sivu määritetään HTML-sivun tapaan, mutta käyttöliittymäkomponentit merkitään käyttämällä erityisiä asp-elementtejä. Kuvassa 9 on esitetty yksinkertaisen ASP.NET-sivun määritys. Sivu sisältää otsikon, kaksi tekstilaatikkoa kuvauksineen ja painonapin. Määrityksen ensimmäisellä rivillä sivu sidotaan ohjelmakoodia sisältävään taustatiedostoon, joka on kirjoitettu C#:lla. Käyttöliittymäkomponentin määritykset sisältävät komponentin tyypin lisäksi yksikäsitteisen tunnisteen (id-attribuutti) ja muita käyttöliittymäkomponentille ominaisia attribuutteja kuten paikan, koon ja ulkoasun määrityksen. Kuvassa 10 on esitetty sama sivu renderöitynä web-selaimelle.

```
<%@ Page language="c#" Codebehind="WebForm1.aspx.cs" AutoEventWireup="false" Inherits="SimpleDemo.WebForm1" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
    <HEAD>
        <title>Simpe Demo</title>
        <meta name="GENERATOR" Content="Microsoft Visual Studio .NET 7.1">
        <meta name="CODE_LANGUAGE" Content="C#">
        <meta name="vs_defaultClientScript" content="JavaScript">
        <meta name="vs_targetSchema" content="http://schemas.microsoft.com/intellisense/ie5">
    </HEAD>
    <body MS_POSITIONING="GridLayout">
        <form id="Form1" method="post" runat="server">
            <asp:TextBox id="tbUsername" style="Z-INDEX: 101; LEFT: 219px; POSITION: absolute; TOP: 141px"
                runat="server" Width="152px" Height="20px"></asp:TextBox>
            <asp:TextBox id="tbPassword" style="Z-INDEX: 104; LEFT: 222px; POSITION: absolute; TOP: 206px"
                runat="server" Width="152px" Height="20px" TextMode="Password"></asp:TextBox>
            <asp:Label id="lblUsername" style="Z-INDEX: 102; LEFT: 215px; POSITION: absolute; TOP: 119px"
                runat="server" Width="75px" Height="21px" Font-Bold="True">Username</asp:Label>
            <asp:Label id="lblPassword" style="Z-INDEX: 103; LEFT: 219px; POSITION: absolute; TOP: 180px"
                runat="server" Width="104px" Height="17px" Font-Bold="True">Password</asp:Label>
            <asp:Button id="cbLogin" style="Z-INDEX: 105; LEFT: 317px; POSITION: absolute; TOP: 247px" runat="server"
                Width="71px" Height="22px" Text="Login"></asp:Button>
            <asp:Label id="lblTitle" style="Z-INDEX: 106; LEFT: 174px; POSITION: absolute; TOP: 64px" runat="server"
                Width="230px" Height="28px" Font-Size="Large" BorderColor="Blue" BorderWidth="1px" BorderStyle="Ridge"
                BackColor="LightCyan">Demo System</asp:Label>
        </form>
    </body>
</HTML>
```

Kuva 9: ASP.NET-sivun määritys.

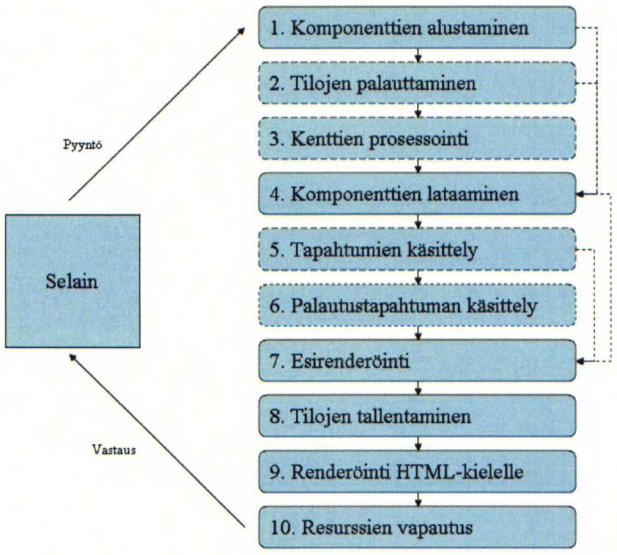


Kuva 10: ASP.NET-sivu renderöitynä selaimelle.

1.5.5.3 ASP.NET-sivujen elinkaari

ASP.NET-käyttöliittymäkomponentit säilyttävät yleensä tilansa sivupyyntöjen välillä, komponenttien tilanhallinnan ymmärtäminen vaatii sivujen elinkaaren tuntemisen. ASP.NET-sivu sisältää aina HTML-lomakkeen, jonka kentät lähetetään takaisin sivulle itselleen. Kun selain pyytää sivua ensimmäisen kerran, se lähetetään vastauksena HTTP GET -pyyntöön. Kun käyttäjä tekee sivulla valintoja, jotka vaativat käsittelyä palvelimella, niin selain lähettää sivun kentät HTTP POST -pyyntönä palvelimelle ja palvelin vastaa tähän lähettämällä saman sivun muokattuna takaisin selaimelle.

ASP.NET-sivujen arkkitehtuuri noudattaa sisäisesti Page Controller -suunnittelumallia [TMQHNL03], sillä sivut käsittelevät saamansa pyynnöt, kutsuvat liiketoimintalogiikkaa ja valitsevat pyyntöön sopivan vastauksen. ASP.NET-sivun elinkaari jakautuu kymmeneen eri vaiheeseen kuten kuvassa 11 on esitetty.



Kuva 11: ASP.NET-sivun elinkaari.

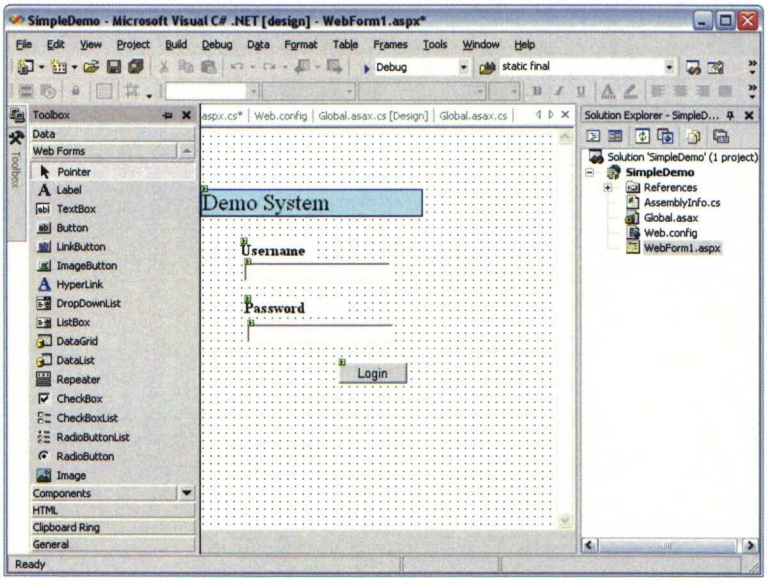
Sivun elinkaari on hieman erilainen riippuen siitä pyydetäänkö sivua ensimmäisen kerran vai onko kyseessä uudelleenlähetys. Kuvaan 11 on merkitty katkoviivoilla ne tilat, jotka koskevat vain sivun uudelleenlähettämistä, sivua ensimmäistä kertaa pyydettäessä nämä tilat ohitetaan. Kun sivupyyntö saapuu, niin aivan aluksi sivun lähdekoodissa määritellyt komponentit ja oliot alustetaan. Jos kyseessä on sivun takaisinlähetys, niin komponenttien vanhat tilat palautetaan lukemalle ne ViewsState-kentästä. ViewState-kenttä on ASP.NETin tapa kuljettaa komponenttien arvoja, se lähetetään oletuksena selaimelle piilotettuna HTML-lomakkeen input-kenttänä. Komponenttien arvoja ei siten tallenneta palvelimelle, vaan asiakkaan päähän selaimeen. ViewState-kenttä sisältää merkkijonopohjaisia avain-arvopareja ja se on Base64-koodattu [Base64-W]. ViewState-kenttä voidaan vaihtoehtoisesti tallentaa web-sovelluksessa palvelimen muistiin tai tietokantaan, mutta se vaatii Page-olion SaveStateToPersistenceMedium- ja LoadPageStateFromPersistence-Medium-metodin korvaamisen omalla toteutuksella. ViewState-kentän käsittelyn jälkeen komponenttien saamat uudet arvot luetaan POST-pyynnön kentistä. Neljännessä vaiheessa komponentit ladataan ja niistä muodostetaan ajonaikainen komponenttipuu, kunkin komponentin latausvaiheeseen liitetty alustuskoodi suoritetaan. Seuraavassa vaiheessa käydään läpi ne komponentit, joiden arvo oli muuttunut, muutos todetaan vertaamalla sivun palautuksessa näkyvää arvoa ViewState-kenttään tallennettuun arvoon. Jos komponentin arvo on muuttunut ja arvon muuttumiselle on määritelty tapahtumankäsittelijä, niin se suoritetaan. Kuudes

vaihe koskee sitä komponenttia, joka aiheutti sivun takaisinlähettämisen, kyseessä voi olla esimerkiksi painonappi tai hyperlinkki. Jos kyseiselle komponentille on määritelty sivun palauttamiseen liittyvä tapahtumankäsittelijä, niin se suoritetaan. Tämä mahdollistaa varsinaisen tapahtumapohjaisen ohjelmoinnin, esimerkiksi napin painallus tai alkion valitseminen listasta tunnistetaan ASP.NET-sovelluksessa tällä menetelmällä. Peruskomponenteista Button-, Calendar-, LinkButton- ja ImageButton -komponentit aiheuttavat aina käyttäjän niitä painaessa sivun takaisinlähettämisen. Muut sivun palautukseen pystyvät peruskomponentit ovat TextBox, CheckBox ja RadioButton sekä ListControl-komponentin aliluokat. Nämä komponentit aiheuttavat takaisinlähetyksen vain jos niiden AutoPostBack-niminen ominaisuus on päällä. Sovelluskehittäjä voi ASP.NET-sivua laatiessaan määrittää, millä komponenteilla AutoPostBack-ominaisuus on käytössä, ominaisuuden arvoa voidaan vaihtaa myös dynaamisesti web-sovellusta suoritettaessa.

Vaiheet seitsemästä kymmeneen liittyvät sivun lähettämiseen selaimelle. Esirenderöintivaiheessa kullekin komponentille annetaan mahdollisuus päivittää tilaansa, tämän vaiheen jälkeen komponenttipuu ja komponenttien arvot kiinnitetään ja tallennetaan ViewState-kenttään. Renderöintivaiheessa sivu ja sen sisältämät komponentit esitetään HTML-kielellä. Muunnoksen jälkeen resurssit vapautetaan ja sivun HTML-koodi lähetetään selaimelle.

1.5.5.4 ASP.NET-kehitystyökalut

ASP.NET-sivuja on mahdollista kirjoittaa pelkällä tekstieditorilla ja kääntää Microsoftin tarjoamilla maksuttomilla kääntäjillä. Käytännössä sivut toteutetaan IDE-työkaluilla, joista suosituin ASP.NET-kehitykseen on Microsoftin Visual Studio.NET 2003. Visual Studio.NET tarjoaa ASP.NET-sivujen kehittämiseen kolme näkymää. Suunnittelunäkymässä ASP.NET-sivu näkyy sellaisena kuin sen tulisi näkyä selaimessa, käyttöliittymäkomponentteja voi vetää komponenttipaletista suoraan sivulle. Toinen näkymä sisältää ASP.NET-sivun kuvauskielisen määrityksen ja se on synkronoitu suunnittelunäkymän kanssa, muutokset toiseen näkymään päivittyvät välittömästi myös toiseen. Kolmas näkymä mahdollistaa ASP.NET-sivun taustakoodin muokkaamisen käytetyn ohjelmointikielen syntaksia ymmärtävällä koodieditorilla. Visual Studio.NET osaa käyttää Microsoftin web-palvelin-ohjelmistoa ASP.NET-sivujen testaamiseen ja virheenjäljittämiseen. Kuvassa 12 on kuvakaappaus Visual Studio.NET 2003 -kehitysympäristön suunnittelunäkymästä.



Kuva 12: Visual Studio.NET 2003 -kehitysympäristön suunnittelunäkymä.

Muitakin kaupallisia kehitysympäristöjä löytyy kuten Borlandin Delphi 2005. Kaupallisten kehitysympäristöjen lisäksi on olemassa myös ilmaisia vaihtoehtoja, joista tunnetuimmat ovat ASP.NET Web Matrix ja SharpDevelop. Ilmaiset vaihtoehdot eivät ainakaan vielä tarjoa yhtä paljon ominaisuuksia kuin kaupalliset kehitysympäristöt.

1.5.6 JavaServer Faces

1.5.6.1 Yleistä

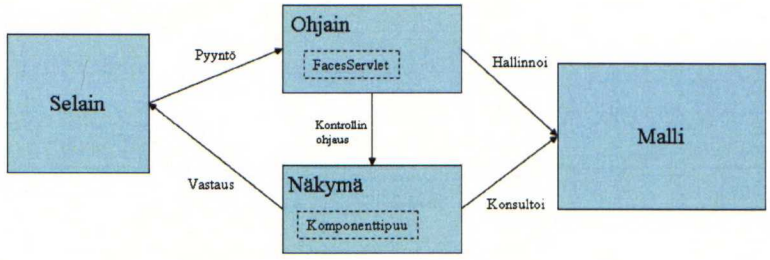
JavaServer Faces (JSF) on komponenttipohjaisten web-käyttöliittymien toteuttamiseen tarkoitettu Java-pohjainen web-sovelluskehys. JavaServer Faces -standardin ensimmäinen versio valmistui Java-yhteisön (eng. Java Community Process) toimesta marraskuussa 2004. Toukokuussa 2004 julkaistiin korjauksia sisältänyt versio 1.1.

JSF on ensimmäinen standardoitu tapa rakentaa käyttöliittymäkomponenttipohjaisia web-sovelluksia Javalla ja se tullaan tulevaisuudessa liittämään osaksi J2EE:tä.

JSF-standardi selkeyttää web-käyttöliittymien rakentamista Java-ympäristöissä, koska tarjolla on ollut useita keskenään epäyhteensopivia sovelluskehyksiä. JSF ei ole syrjäyttämässä Servlet- tai JSP-teknologioita vaan se rakentuu osin niiden päälle. JSF-ohjelmia voidaan ajaa missä tahansa Java-pohjaisessa web-palvelimessa tai Servlet-säiliössä mikäli se tukee vähintään Servlet-spesifikaation versiota 2.3 ja JSP-spesifikaation versiota 1.2. Näiden vaatimuksien lisäksi JSF-sovelluksen ajamiseen tarvitaan JSF-toteutus. Sun Microsystems tarjoaa maksuttoman referenssitoteutuksen ja useat valmistajat ovat tuomassa omia toteutuksiaan markkinoille. Saatavilla on myös avoimen lähdekoodin yhteisöjen toteutuksia, joista tunnetuin on Apache MyFaces.

JSF noudattaa MVC-suunnittelumallia (eng. Model-View-Controller) [MVC-W]. MVC-mallin voidaan katsoa olevan arkkitehtuurimalli, koska se jakaa sovelluksen kolmeen toisistaan mahdollisimman riippumattomaan osaan. Osista käytetään nimityksiä malli, ohjain ja näkymä. Web-sovellusten yhteydessä puhutaan MVC Model 2 -mallista, jossa käytettävien protokollien ja kuvauskielten asettamat rajoitukset on huomioitu [Man04]. MVC-mallissa malli on ohjelmistotekninen esitys ratkottavasta ongelmakentästä ja se sisältää tietoa ja logiikkaa. Näkymä tarjoaa käyttäjälle esityksen mallista ja ohjain hallitsee käyttäjän vuorovaikutusta mallin kanssa [DLWM04]. Kuvassa 13 on esitetty yleisellä tasolla JSF-sovelluksen rakenne, joka noudattaa MVC-mallia. Kun selain pyytää JSF-sivua, niin pyyntö kulkee ohjaimen kautta. Ohjain on toteutettu Java-servlettinä ja siitä käytetään nimitystä FacesServlet. Ohjain käsittelee pyynnön, ylläpitää mallia ja valitsee vastaukseen liitettävän näkymän. Näkymä sisältää käyttöliittymäkomponenttipuun, joka esitetään

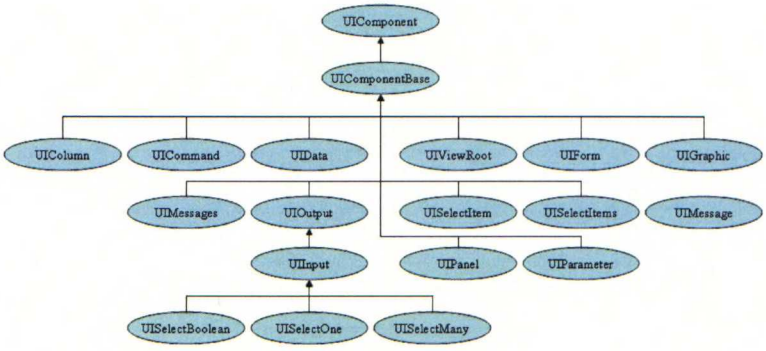
selaimelle sopivalla esitystekniikalla. Malli koostuu Java-olioista tai EJB-pavuista. Koska FacesServlet-ohjain käsittelee kaikki selaimelta tulevat pyynnöt, se toteuttaa Front Controller -ohjainmallin [DLWM04].



Kuva 13: JSF-sovellus noudattaa MVC-mallia.

1.5.6.2 JSF-käyttöliittymäkomponentit

JSF on toteutettu siten, että se on riippumaton käytettävästä esityskerroksesta. JSF-sovellus voi siten renderöidä näkymiä millä tahansa esitystekniikalla kuvauskielistä vektorigrafiikkaan. Käytännössä suosituin esitystekniikkaa on HTML 4.0.1 ja jokaisen JSF-toteutuksen pitää tukea renderöintiä HTML-kielelle. JSF-standardi määrittelee joukon peruskomponentteja, joiden hierarkia on esitetty kuvassa 14. Standardissa peruskomponentit on esitetty käsitteellisellä tasolla, eikä niiden ulkoasuun ole otettu kantaa. Esimerkiksi UIGraphic-komponentti on tarkoitettu graafisen kuvan esittämiseen ja UICommand-komponentti soveltuu komentojen antamiseen sovellukselle. Jotta komponentit voidaan näyttää käyttäjälle, tarvitaan renderöintikirjasto. HTML-renderöintikirjasto sisältää 24 komponenttia, joilla web-käyttöliitymiä voidaan rakentaa. HTML-komponentit sisältävät tavallisimmat käyttöliittymäelementit kuten tekstilaatikot, valintalistat, painonapit ja hyperlinkit. Perustoiminnallisuuksien lisäksi HTML-komponttien joukosta löytyy UIData-pohjainen komponentti, jolla voidaan esittää tietoa taulukkomuodossa.



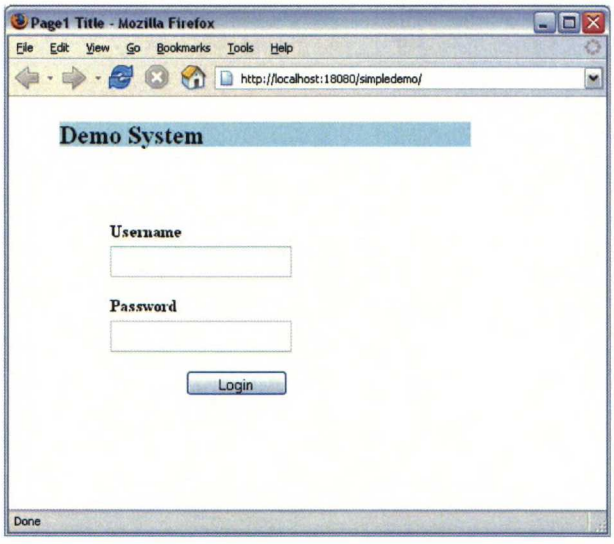
Kuva 14: JSF-peruskomponenttien hierarkia.

Koska jokaisen JSF-toteutuksen on tuettava HTML-renderöintiä ja HTML on yleisin web-selaimissa toimiva kuvauskieli, keskitytään tässä yhteydessä JSF-sivujen määrittämiseen HTML-renderöintikirjastolla.

Käytettäessä HTML-renderöintikirjastoa JSF-sivu toteutetaan JSP-sivuna, johon on lisätty erityisiä JSF-elementtejä. Tämän lisäksi tarvitaan avustava papu (eng. backing bean), johon JSF-elementeissä viitataan. Avustava papu on Java-luokka, johon voidaan sisällyttää tapahtumien käsittelyyn tarkoitettuja metodeja sekä julkisia ominaisuuksia. JSF-elementeillä määritetään sivuun sisällytettävät käyttöliittymäkomponentit, kullekin komponentille annetaan yksikäsitteinen tunniste. Käyttöliittymäkomponentit sidotaan avustaviin papuihin käyttämällä JSF EL-kieltä (JavaServer Faces Expression Language). EL-kieli on määritelty JSF-spesifikaation versiossa 1.1 ja on syntaksiltaan samanlainen kuin JavaServer Pages 2.0 -spesifikaatiossa määritetty ilmaisukieli [JSF-1.1] [JSP-2.0]. JSF-spesifikaation tulevassa versiossa 1.2 ilmaisukieleksi vaihdetaan JSP-spesifikaation versiossa 2.1 määritetty yhtenäistetty ilmaisukieli (unified Expression Language) [JSF-1.2] [JSP-2.1]. Kuvassa 15 on esitetty yksinkertaisen JSF-sivun määritys. Sivu sisältää otsikon, kaksi tekstikenttää ja painonapin. Sivun käyttöliittymäkomponentit on sidottu "Page1"-nimiseen avustavaan papuun. Käyttöliittymäkomponenttien ulkoasut on kuvattu CSS-tyylimäärityksillä [CSS-W]. Kuvassa 16 on esitetty miltä sivu näyttää selaimessa.

```
<?xml version="1.0" encoding="UTF-8"?>
<jsp:root version="1.2" xmlns:f="http://java.sun.com/jsf/core" xmlns:h="http://java.sun.com/jsf/html"
    xmlns:jsp="http://java.sun.com/JSP/Page">
    <jsp:directive.page contentType="text/html;charset=UTF-8" pageEncoding="UTF-8"/>
    <jsp:text><![CDATA[
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
]]></jsp:text>
    <f:view>
        <html lang="fi-FI" xml:lang="fi-FI">
            <head>
                <meta content="no-cache" http-equiv="Cache-Control"/>
                <meta content="no-cache" http-equiv="Pragma"/>
                <title>Page1 Title</title>
                <link href="resources/stylesheet.css" rel="stylesheet" type="text/css"/>
            </head>
            <body style="-rave-layout: grid">
                <h:form binding="#{Page1.form1}" id="form1">
                    <h:outputText binding="#{Page1.outputText1}" id="outputText1"
                        style="background-color: rgb(204, 255, 255);
                        font-size: 24px; font-weight: bold; height: 48px; left: 96px; top: 120px;
                        position: absolute; width: 388px" value="Demo System"/>
                    <h:outputText binding="#{Page1.outputTextUsername}" id="outputTextUsername"
                        style="height: 24px; position: absolute; width: 168px; font-weight: bold;
                        left: 144px; top: 216px" value="Username"/>
                    <h:inputText binding="#{Page1.textFieldUsername}" id="textFieldUsername"
                        style="height: 24px; left: 144px;top: 240px; position: absolute; width: 168px"/>
                    <h:outputText binding="#{Page1.outputTextPassword}" id="outputTextPassword"
                        style="font-weight: bold; height: 24px; left: 144px; top: 288px;
                        position: absolute; width: 168px" value="Password"/>
                    <h:inputText binding="#{Page1.textFieldPassword}" id="textFieldPassword"
                        style="height: 24px; left: 144px; top: 312px; position: absolute; width: 168px"/>
                    <h:commandButton action="#{Page1.buttonLogin_action}" binding="#{Page1.buttonLogin}" id="buttonLogin"
                        style="height: 24px; left: 216px; top: 360px; position: absolute; width: 96px" value="Login"/>
                </h:form>
            </body>
        </html>
    </f:view>
</jsp:root>
```

Kuva 15: JSF-sivun määritys.



Kuva 16: JSF-sivu renderöitynä selaimelle.

EL-kieli mahdollistaa käyttöliittymäkomponentin näyttämän arvon sitomisen avustavan pavun ominaisuuteen. Esimerkiksi web-sivulla oleva sähköpostiosoitteen syöttämiseen tarkoitetun tekstikentän arvo voidaan sitoa avustavan pavun String-tyyppiseen nimettyyn ominaisuuteen. Tällöin tekstikentän arvo on synkronoitu pavun ominaisuuden kanssa. Jos käyttäjä muokkaa sähköpostiosoitetta web-sivulla, niin pavun ominaisuus päivittyy vastaamaan sitä arvoa heti seuraavalla kerralla kun sivu

lähetetään palvelimelle. Vastaavasti jos sähköpostiosoitetta muokataan pavussa, niin muutos näkyy myös web-sivulla. Web-sivulla näytettävä data on yleensä tekstipohjaista, mutta avustavassa pavussa käytetään olioita tai primitiivityyppejä. Esimerkiksi päivämäärä voidaan esittää web-sivulla monessa eri muodossa, mutta avustavassa pavussa se on yleensä Javan Date-luokan ilmentymä. Jotta synkronointi olisi mahdollista, JSF tarjoaa konverttereita, joilla olioita voidaan muuntaa merkkijonoiksi ja toisin päin. JSF sisältää joukon peruskonverttereita, joilla voidaan konvertoida numeroita ja päivämääriä. Konvertterit voidaan liittää käyttöliittymäkomponentteihin joko ohjelmakoodissa tai staattisesti EL-kielellä komponentin JSF-elementissä. Uusia konverttereita on mahdollista rakentaa määrittelemällä luokka, joka toteuttaa Converter-rajapinnan.

Käyttöliittymäkomponentti voidaan sitoa avustavassa pavussa määriteltyyn käyttöliittymäkomponentin instanssiin, tällöin käyttöliittymäkomponenttia voidaan muokata ohjelmallisesti avustavassa pavussa.

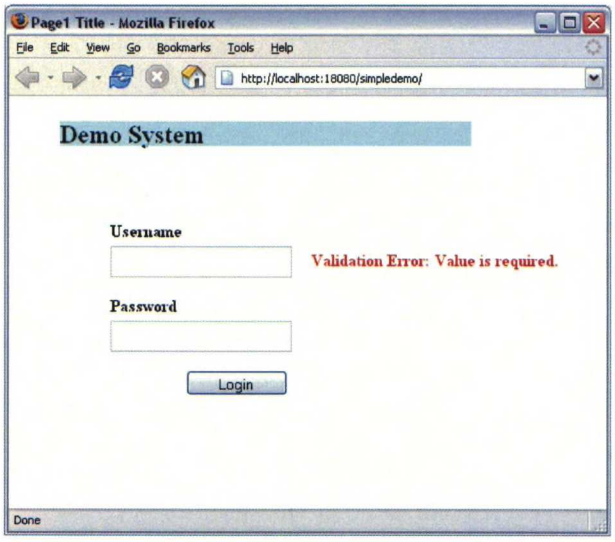
Tapahtumienkäsittelijät määritetään metodeina avustavissa pavuissa ja käyttöliittymäkomponenttien aiheuttamat tapahtumat sidotaan näihin metodeihin käyttämällä EL-kieltä. JSF-komponentit tukevat kahdenlaisia tapahtumia.

UIInput-pohjaiset komponentit on tarkoitettu tietojen syöttämiseen ja niille voidaan määrittää arvon muutokseen liittyvä tapahtumankäsittelijä. Tapahtumankäsittelijän määrittäminen tehdään merkitsemällä komponentin JSF-elementtiin EL-kielellä viittaus avustavan pavun tapahtumankäsittelijämetodiin. Samalla sivulla olevat komponentit voivat käyttää eri papuja ja usealla sivulla voi olla viittaus samaan papuun. UICommand-pohjaisiin komponentteihin voidaan liittää tapahtumankäsittelijä, joka liittyy komennon antamiseen sovellukselle. Esimerkiksi napin tai hyperlinkin painaminen aiheuttaa tällöin tapahtumankäsittelijän suorittamisen. Tyypillisesti web-palvelin suorittaa tapahtumankäsittelijän ja sivu näytetään uudestaan selaimelle. UICommand-pohjaisten komponenttien voidaan määrittää aiheuttavan tapahtumia, jotka ovat sidoksissa JSF:n navigointi-järjestelmään. Tällainen navigointitapahtuma voi olla staattinen tai dynaaminen. Staattinen tapahtuma antaa vakio navigointikomennon kun dynaamisessa tapahtumassa voidaan suorittaa ensin logiikkaa ja sen pohjalta päättää mikä navigointikomento annetaan.

UIInput-pohjaisten komponenttien arvoja voidaan tarkastaa validaattoreilla. Arvojen tarkastaminen voidaan jättää komponentin suoritettavaksi tai suorittaa komponentin ulkopuolella. Komponentin sisällä tehty tarkastus vaatii validate-

metodin korvaamisen. Komponentin ulkopuolella tehty tarkastus voidaan toteuttaa Validator-rajapinnan toteuttavilla komponenteilla tai delegoimalla tarkastus esimerkiksi pavulle. JSF-spesifikaatio määrittää kolme Validator-komponenttia, joilla voidaan tarkastaa syötteen pituus tai onko syöte lukuna annettujen arvojen sisällä. Validaattoreita voi rakentaa itse toteuttamalla Validator-rajapinnan. Tarkastuksen delegointi pavulle tehdään määrittämällä komponentin JSF-elementtiin viittaus pavun validointimetodiin. Validaattorin voi myös rekisteröidä dynaamisesti ohjelmakoodissa.

UIMessage- ja UIMessages-komponentit liittyvät JSF:n viestimekanismiin. Viestit voivat liittyä yksittäisiin komponentteihin tai koko komponenttipuuhun. UIMessage-komponentti soveltuu yhden käyttöliittymäkomponentin viestien näyttämiseen kun UIMessages-komponentilla voidaan esittää kaikki pyynnön käsittelyssä syntyneet viestit. Tyypillisesti viestejä käytetään validaattoreiden ja konvertterien aiheuttamien virheilmoitusten näyttämiseen. Esimerkiksi tekstikentän arvoon voidaan liittää syötteen pituuden tarkastava validaattori. Jos syöte ei ole kelvollinen, niin validaattori tuottaa virheviestin. UIMessage-komponentti voidaan sitoa tekstikentän arvoon, jolloin se näyttää tekstikentän virheilmoitukset kuten kuvassa 17 on havainnollistettu. Ohjelmakoodissa viestejä edustaa FacesMessage-luokka. Viestit jakautuvat neljään eri tasoon. Alimman tason viestit ovat tiedotuksia ja toisen tason viestit ovat varoituksia. Kolmannen ja neljännen tason viestit ovat virheviestejä, joista jälkimmäiset kriittisyydeltään erittäin vakavia. Kuhunkin viestiin voidaan liittää tiivistelmä ja tarkempi seloste.

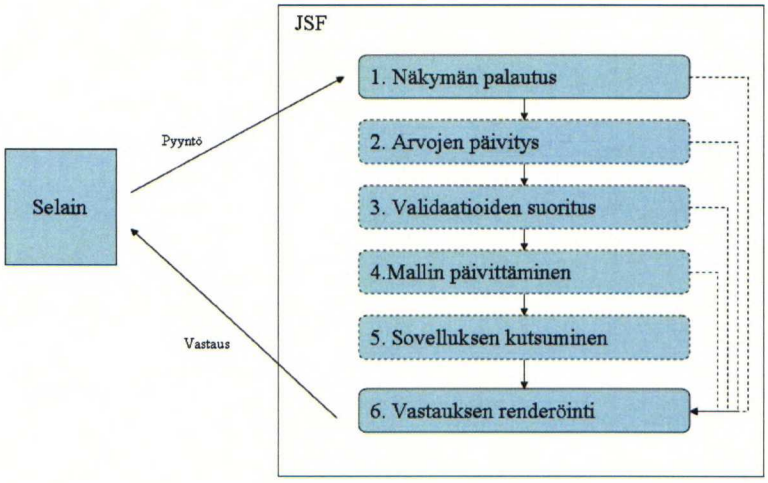


Kuva 17: Esimerkki validaattorin tuottamasta virheilmoituksesta.

JSF-sovelluksen sivujen välinen navigointi määritetään erillisellä XML-tiedostolla. XML-tiedostoon määritetään navigointisääntöjä, jotka kertovat millä navigointi-komennolla siirrytään millekin sivulle. Navigointikomennot ovat vapaasti valittavia merkkijonoja.

1.5.6.3 JSF-sivujen elinkaari

JSF-toteutus käsittelee selaimen lähettämän pyynnön yleensä kuusivaiheisesti. Päävaiheet on esitetty kuvassa 18, katkoviivalla merkittyjä vaiheita ei välttämättä aina suoriteta. Kun JSF saa selaimelta sivupyynnön, niin se tarkistaa aluksi löytyykö pyydettävälle sivulle jo tallennettu komponenttipuu. Mikäli komponenttipuu löytyy, niin se ladataan ja palautetaan pyyntöä edeltäneeseen tilaan. Jos komponenttipuuta ei löydy, se luodaan JSF-sivun määrityksen pohjalta. Jos sivupyyntö ei sisällä välitettyjä parametreja, niin suoritus siirtyy suoraan viimeiseen vaiheeseen, muutoin siirrytään vaiheeseen kaksi.



Kuva 18: JSF-sivupyynnön käsittelyn päävaiheet.

Vaiheessa kaksi komponenttipuussa oleville käyttöliittymäkomponenteille annetaan mahdollisuus päivittää arvonsa sivupyynnön sisältämien parametrien mukaisesti. Parametrien lukemisessa hyödynnetään konverttereita niiden komponenttien osalta, joille konversioita on määritetty. Mikäli konversiossa tapahtuu virheitä, siirrytään suoraan vaiheeseen kuusi. Ne komponentit, joiden arvot ovat muuttuneet, lisäävät muutostapahtuman JSF-toteutuksen tapahtumajonoon. Tässä vaiheessa tunnistetaan sivupyynnön aiheuttanut UICommand-pohjainen komponentti ja tapahtumajonoon lisätään tämän komponentin aiheuttama komentotapahtuma. Jos komponentteihin on liitetty validaattoreita, niin ne suoritetaan vaiheessa kolme. Mikäli validaatio aiheuttaa virheen, siirrytään vaiheeseen kuusi.

Vaiheessa neljä käyttöliittymäkomponenttien sidokset avustaviin papuihin käydään lävitse ja papujen arvot päivitetään vastaamaan komponenttien uusia arvoja.

Mikäli mallin päivittämiseen on sidottu tapahtumankäsittelijöitä, ne voivat keskeyttää koko sivupyynnön käsittelyn, jolloin siirrytään viimeiseen vaiheeseen.

Viidennessä vaiheessa käsitellään ensimmäisessä vaiheessa tunnistettu komento-tapahtuma. JSF-toteutus purkaa tämän tapahtuman tapahtumajonosta ja tiedottaa asiasta kaikille rekisteröidyille tapahtumankäsittelijöille, jotka sitten suoritetaan. Tässä vaiheessa tapahtumankäsittelijät tyypillisesti aktivoivat varsinainen liiketoimintalogiikan suorituksen.

Navigointijärjestelmään sidotut tapahtumankäsittelijät pystyvät vaikuttamaan siihen mikä sivu selaimelle lähetetään vastauksena sivupyyntöön. Jos yksikään tapahtuman-käsittelijä ei anna navigointikomentoa, niin suoritus etenee viimeiseen vaiheeseen.

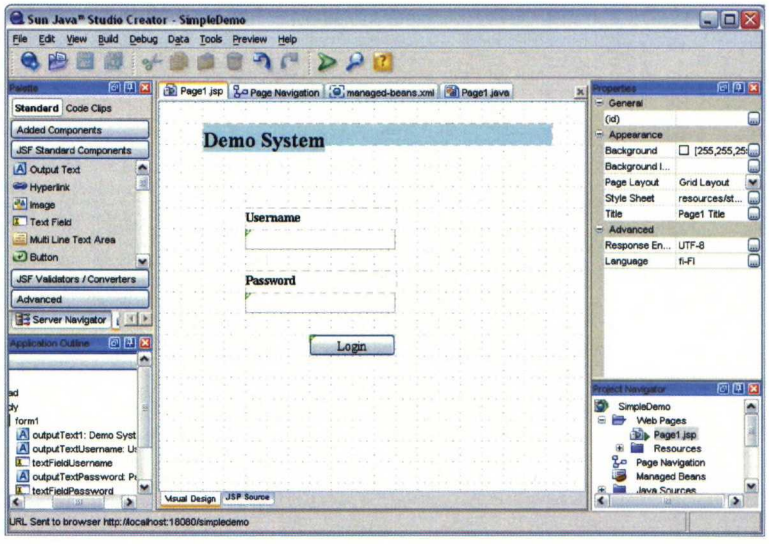
Mikäli navigointijärjestelmä ohjaa toiselle JSF-sivulle, suoritetaan sen sivun osalta sama pyynnönkäsittelyketju, mutta tyypillisesti vain vaiheet yksi ja kuusi. JSF-sivupyynnön viimeisessä vaiheessa komponenttipuu renderöidään selaimelle sopivaan muotoon käyttämällä sovellukseen rekisteröityä renderöintikirjastoa. Komponenttipuun tila tallennetaan joko palvelimelle tai selaimelle lähetettävään vastaukseen. Palvelinta käytettäessä yleisin sijoituspaikka on istunto-olio.

JSF tukee mukautettujen komponenttien valmistamista. Mukautettuja komponentteja voi luoda itse tai hankkia kolmansilta osapuolilta. Mukautettu komponentti voidaan rakentaa monella eri tavalla, yksi tapa on yhdistää olemassa olevia komponentteja yhdeksi isommaksi koostekomponentiksi. Mukautettu komponentti voidaan muodostaa periyttämällä UIComponentBase-luokka tai jokin sen aliluokka. Työläin tapa luoda mukautettu komponentti on periä abstrakti UIComponent-luokka. Jotta mukautetun komponentin voisi sisällyttää JSP-sivulle, on sitä varten luotava tai laajennettava olemassa olevaa elementtikirjastoa. Komponentti saa tällöin oman JSF-elementin, jota käytetään JSF-sivun määrityksessä. Jos JSF-käyttöliittymäkomponentin ulkoasuun halutaan lisätä uusia ominaisuuksia tai tukea uudenlaisia asiakasohjelmia, niin mukautetun komponentin luominen ei ole välttämätöntä vaan tällöin voidaan tehdä mukautettu renderöijä. Mukautettujen renderöijien käyttö vaatii myös uusien JSF-elementin määrittämisen.

1.5.6.4 JSF-kehitystyökalut

JSF-sivujen tekeminen onnistuu millä tahansa tekstieditorilla, mutta graafisen kehitysympäristön käyttö nopeuttaa sivujen toteuttamista merkittävästi. JSF-standardia suunniteltaessa pyrittiin hyvään yhteensopivuuteen WYSIWYG-kehitysympäristöjen [WYSIWYG-W] kanssa ja moni valmistaja onkin jo lisännyt ohjelmistoihinsa tuen JSF:lle. Sivujen suunnittelun lisäksi kehitysympäristöt tarjoavat graafiset työkalut navigointisääntöjen rakentamiseen ja konfiguraatiotiedostojen editointiin. Tunnetuimmat JSF-kehitykseen sopivat kaupalliset graafiset kehitysympäristöt ovat Sun Microsystemsin Java Studio Creator, IBM:n Websphere ja Oraclen JDeveloper. Jotkin kehitysympäristöt eivät tarjoa sivujen tekemiseen graafista suunnittelunäkymää, mutta avustavat JSF-elementtien kirjoittamisessa (esimerkiksi Borlandin JBuilder 2005). Esimerkki Java Studio Creatorin suunnittelunäkymästä on esitetty kuvassa 19. Avoimen lähdekoodin yhteisöiltä ei löydy vielä WYSIWYG-editoria JSF-kehitykseen, mutta tilanne saattaa muuttua kun

JSF yleistyy. Eräs suosittu avoin kehitysympäristö on Eclipse ja siihen on saatavilla JSF-kehitystä helpottavia lisämoduuleita.



Kuva 19: Sun Java Studio Creatorin suunnittelunäkymä.

1.6 Yhteenveto

Tässä luvussa käsiteltiin web-sovelluskehityksen historiaa, vanhempia web-teknologioita sekä uusia komponenttipohjaisia teknologioita. Web-sovelluskehityksen ongelmat juontuvat siitä, että web-sovellukset toimivat tilattoman HTTP-protokollan päällä, joka aikoinaan suunniteltiin staattisen tiedon jakamiseen. Nykyisin web-sovellusten käyttöliittymiltä vaaditaan yhä enemmän, usein jopa ominaisuuksia, jotka aiemmin löytyivät vain työpöytäsovelluksista. Web-sovellusten on näytettävä aiempaa enemmän dynaamista aineistoa, joka usein haetaan tietokannasta tai muusta tietolähteestä.

Ensimmäisen sukupolven web-teknologiat kuten CGI ja JavaServlets toimivat hyvin lähellä HTTP-protokollaa ja niitä käytettiin yleensä vain dynaamisen aineiston lisäämiseen web-sivulle tai tiedon vastaanottamiseen käyttäjiltä. Toisen sukupolven web-teknologiat kuten ASP, JSP ja PHP ovat dokumenttipohjaisia ja niillä voidaan rakentaa kokonaisia sivustoja. Näiden teknologioiden ongelmana on se, että määrittelydokumentit sisältävät sekä sivun visuaaliseen ulkoasuun vaikuttavia elementtejä että dynaamisen aineiston ja käyttöliittymälogiikan vaatimaa ohjelmakoodia. Tällaisten määrittelydokumenttien ylläpitäminen on osoittautunut hankalaksi.

Kolmannen sukupolven web-teknologiat kuten ASP.NET ja JavaServer Faces perustuvat käyttöliittymäkomponentteihin. Käyttöliittymäkomponentilla on visuaalinen ulkoasu ja sen tehtävä on vuorovaikuttaa web-sivulla käyttäjän kanssa. Käyttöliittymäkomponenttiteknologioilla web-sivujen käyttöliittymiä voidaan rakentaa niin, että sovelluskehittäjän ei tarvitse huolehtia kuvauskielisistä määrityksistä, vaan niiden tuottaminen on järjestelmän vastuulla. Käyttöliittymäkomponenttiteknologiat pystyvät ottamaan huomioon käytetyn selainlaitteen ominaisuudet ja rajoitukset. Komponenttiteknologiat sisältävät joukon peruskomponentteja tavallisimpiin käyttötilanteisiin, mutta mahdollisuus rakentaa tai ostaa uusia komponentteja korottaa näiden teknologioiden käyttöarvoa.

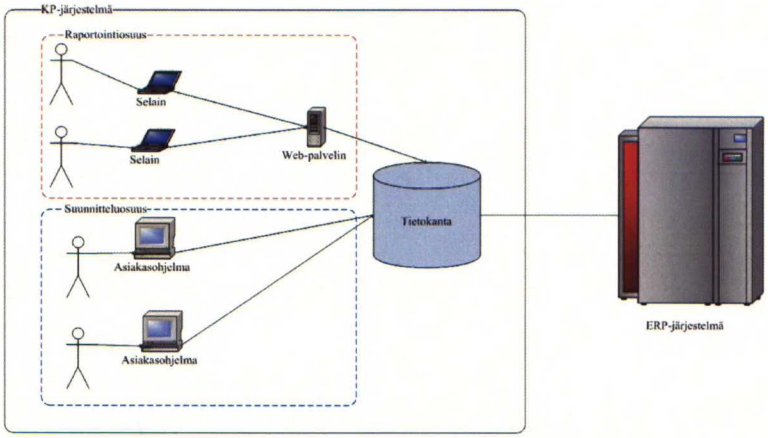
2 KP-järjestelmän raportointiosuus

2.1 Johdanto

Tässä luvussa esitellään aluksi Nokialla tuotantokäytössä oleva KP-järjestelmä ja sen raportointiosuus, sitten käydään läpi raportointiosuuden nykyinen toteutus ja siihen liittyvät ongelmat. Nykyisen toteutuksen ongelmat pyritään ratkaisemaan käyttöliittymäkomponenttiteknologioilla, minkä johdosta on rakennettu kaksi varsin pitkälle kehitettyä prototyyppiä. Luvun loppupuolella esitellään ASP.NET- ja JSF-prototyyppien suunnitteluperiaatteet, arkkitehtuurit ja käytetyt kehitystyökalut.

2.2 Yleiskuvaus järjestelmästä

KP-järjestelmä on tarkoitettu kustannuspaikkojen menojen kuukausittaiseen suunnitteluun ja seurantaan. Toimintakulujen ja investointien lisäksi myös henkilöstömäärät ovat suunniteltavissa. Käyttäjät syöttävät järjestelmään ennusteita ja toteutumat tuodaan yrityksen ERP-järjestelmästä. KP-järjestelmällä on yli 5000 käyttäjää ympäri maailmaa. Kuvassa 20 on esitetty KP-järjestelmän arkkitehtuuri, järjestelmän ytimenä on tietokanta, johon arvioidut kustannukset kirjataan. Tietokantaan tuodaan joka kuukausi ERP-järjestelmästä toteutuneet kustannukset, samalla ERP-järjestelmään siirretään tietokantaan tallennetut ennusteet. Tietokannan lisäksi järjestelmä sisältää kaksi alijärjestelmää, jotka ovat raportointi- ja suunnitteluosuus.



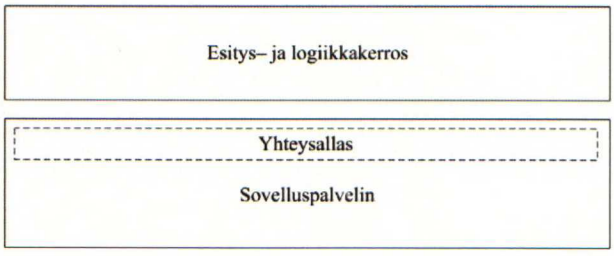
Kuva 20: KP-järjestelmän arkkitehtuuri.

Suunnitteluosuus sisältää asiakasohjelman, jonka kautta käyttäjät tekevät kuukausittain uudet suunnitelmat. Asiakasohjelma on graafinen työpöytäohjelma, jota ajetaan käyttäjän koneessa. Kustannuspaikkojen johtajat ja muut tiedoista kiinnostuneet voivat seurata kustannuksia käyttämällä järjestelmän raportointiosuutta. Raportointiosuus on web-pohjainen sovellus ja sitä käytetään selaimen kautta. Ohjelma hakee tiedot raportteihin tietokannasta. Kuvassa 20 raportointiosuus on merkitty katkoviivoilla. Raportointiosuuden käyttäjät pystyvät katsomaan erilaisia raportteja niiltä kustannuspaikoilta, joihin heillä on oikeudet. Raportit voivat koskea yhtä tai useampaa kustannuspaikkaa ja sisältää sekä toteutumia että ennusteita. Käyttäjät voivat valita haluamansa kohdevaluutan, jolloin sovellus suorittaa valuuttakonversion ennen raportin näyttämistä.

2.3 Raportointiosuuden nykyinen toteutus

2.3.1 Toteutuksen arkkitehtuuri

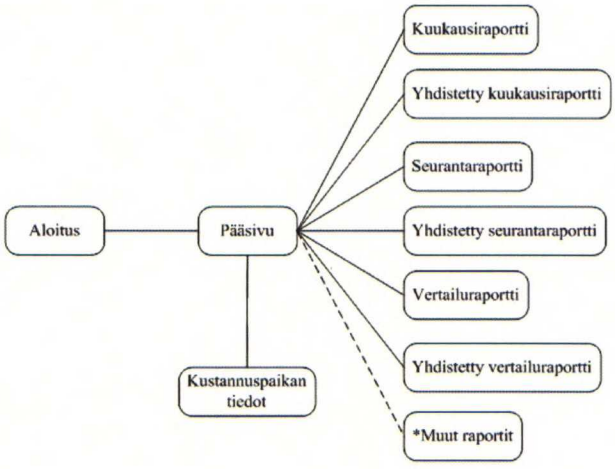
KP-järjestelmän raportointiosuus on toteutettu JSP-sivustona ja sitä ajetaan Java-pohjaisella sovelluspalvelimella. Arkkitehtuuriltaan toteutus nojaa vahvasti JSP-sivuihin, sillä ne sisältävät sekä esitys- että logiikkakerroksen kuten kuvassa 21 on esitetty. Sovellus hakee tarvitsemansa tiedot tietokannasta lähettämällä JDBC-rajapinnan kautta SQL-kyselyjä. Koska tietokantayhteyksien avaaminen ja sulkeminen kuluttaa resursseja, toteutus käyttää sovelluspalvelimen tarjoamaa yhteysallasta. Yhteysallas sisältää joukon avoimia yhteyksiä tietokantaan, yhteyksien lukumäärää kasvatetaan ja pienennetään riippuen järjestelmän kuormasta.



Kuva 21: Raportointiosuuden nykyisen toteutuksen arkkitehtuuri.

Sivuston rakenne on esitetty kuvassa 22. Aloitussivulla käyttäjä tunnistetaan, jonka jälkeen siirrytään pääsivulle. Pääsivulta käyttäjä pystyy valitsemaan raportin tyypin, kustannuspaikat ja kohdevaluutan, sekä käynnistämään raportin luonnin. Jokaista raporttityyppiä vastaa tietty JSP-sivu. Jos raporttiin halutaan useamman kuin

yhden kustannuspaikan luvut, niin kyseessä on yhdistetty raportti, jolloin kustannukset summataan. Raporttisivulla kustannukset näytetään kategorioittain, käyttäjä voi valita näytetäänkö kustannusten yksityiskohtaiset rivit vai ei. Pääsivulta voi avata yksittäisen kustannuspaikan tiedot ja ne näytetään tällöin omalla sivullaan.



Kuva 22: Raportointiosuuden sivurakenne.

2.3.2 Toteutuksen ongelmat

Nykyisen raportointiosuuden ylläpito on osoittautunut hankalaksi. Toteutukseen kohdistuu toistuvia muutospaineita kuten uudentyyppisten raporttien lisäämistä tai vanhojen raporttien muuttamista. Muutosten tekeminen ei ole helppoa, koska toteutuksen esityskerrosta ja logiikkakerrosta ei ole erotettu toisistaan, vaan JSP-sivut sisältävät mm. ohjelmakoodia, jolla kommunikoidaan tietokannan kanssa. Kun raporttien ulkoasua halutaan muokata, niin on varottava, ettei raporttien sisältö muutu. Vastaavasti jos raportteihin halutaan tehdä sisäisiä muutoksia, niin ne voivat vaikuttaa ulkoasuun. Ratkottavaa ongelmakenttää ei ole mallinnettu oliolla, vaan tieto haetaan sivuille suoraan tietokannasta, tämä aiheuttaa riippuvuutta tietokannan rakenteesta. Uusien raporttien lisääminen on työlästä, koska olemassa olevia raportteja ei voi juurikaan hyödyntää. Ylläpidon tarvetta lisää myös se, että yhden kustannuspaikan ja useamman kustannuspaikan sisältävät raportit luodaan eri sivuilla.

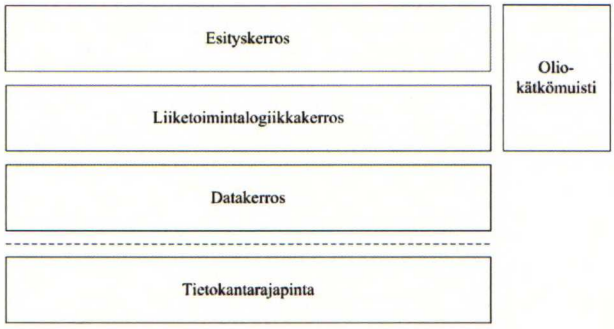
2.4 Uudet komponenttipohjaiset toteutukset

Tässä työssä rakennettiin kaksi käyttöliittymäkomponenttipohjaista prototyyppiä, jotka toteuttavat KP-järjestelmän raportointiosuuden. Toinen prototyyppi tehtiin ASP.NETillä ja toinen JSF:llä. Prototyyppien suunnittelussa piti huomioida useita vaatimuksia, jotka on esitetty seuraavassa listassa:

- Prototyypin pitää olla jatkokehitettävissä tuotantokelpoiseksi järjestelmäksi.
- Uusien raporttien lisäämisen tulee olla helppoa.
- Valuuttakonversio on suoritettava sovelluksessa eikä tietokannassa.
- Prototyypin ulkoasun pitää näyttää mahdollisimman samalta kuin nykyisen toteutuksen.
- Prototyypin tulee toimia yleisimmillä vähintään HTML:n versiota 4.0 tukevilla selaimilla (Internet Explorer, Mozilla, Opera ja Firefox).
- Prototyypin on oltava vähintään yhtä tehokas kuin nykyinen toteutus.
- Prototyypin on skaalauduttava vähintään yhtä hyvin kuin nykyisen toteutuksen.
- Tiedon kulku prototyypin ja tietokannan välillä tulisi olla mahdollisimman tehokasta, tietokantaa pitää rasittaa mahdollisimman vähän.
- Prototyypin pitää noudattaa kolmikerrosarkkitehtuuria, jossa sovellus jaetaan esitys-, logiikka-, ja datakerrokseen.
- Esityskerros ei saa sisältää yhtään liiketoimintalogiikkaa.
- Esityskerroksessa HTML-kielen tai muun kuvauskielen tuottaminen tulee jättää käyttöliittymäkomponenttien vastuulle niin pitkälle kuin mahdollista.
- Logiikkakerros on mallinnettava olioilla.
- Logiikkakerroksen tulee olla täysin riippumaton käytetystä tietokannasta.
- Tietokannan kanssa tulee keskustella vain käyttämällä tallennettuja proseduureja.

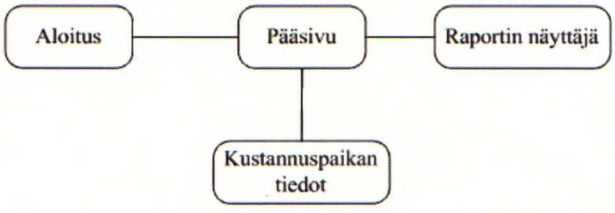
Prototyypit päätettiin toteuttaa niin, että ne käyttävät saamaa koodirunkoa mahdollisimman paljon. Näin prototyyppien vertailu saadaan tasapuoliseksi. Prototyyppien eri kerroksista laadittiin ohjelmointikieliriippumattomat UML-luokkakaaviot ja tärkeimmistä tilanteista UML-sekvenssikaaviot. Prototyyppien arkkitehtuuri sisältää kolme kerrosta kuten kuvassa 23 on esitetty. Prototyypit ovat datakerrokselta ja liiketoimintalogiikkakerrokselta lähes identtiset, ainoastaan käytettyjen ohjelmointikielten ominaisuudet ja luokkakirjastot aiheuttivat pieniä

eroavaisuuksia. Molemmat kerrokset toteutettiin käyttämällä tavallisia luokkia (eng. Plain Old Java Objects, Plain Old .NET Objects). Prototyyppien esityskerrokset on mukautettu käytetyn komponenttiteknologian mukaisiksi, mutta molempien prototyyppien sivuhierarkia ja liitännät liiketoimintalogiikkakerrokseen ovat yhtenevät.



Kuva 23: Prototyyppien arkkitehtuuri.

Prototyyppien käyttöliittymät eli web-sivut muodostettiin käyttöliittymäkomponenteilla eikä HTML-kieltä tarvinnut itse tuottaa. Sivujen rakenne on esitetty kuvassa 24. Aloitussivu pysyi visuaaliselta ulkoasultaan hyvin samanlaisena kuin alkuperäisessä toteutuksessa, samoin pääsivu ja sivu, jolta voi katsoa kustannuspaikkojen tietoja.



Kuva 24: Prototyyppien sivurakenne.

Raporttien näyttämiseen käytetään vain yhtä sivua. Raportin näyttäjälle annetaan Raportti-tyyppinen olio, jonka se käy lävitse ja esittää graafisesti taulukkomuodossa. Raportti voi sisältää yhden tai useamman kustannuspaikan luvut. Yhden sivun käyttäminen on mahdollista, koska liiketoimintakerroksessa raporteille on määritelty oliohierarkia ja Raportti-olio on kaikkien raporttien kantaluokka. Raportin näyttäjää voi pitää yhtenä suurena komponenttina, joka on erikoistunut raporttien esittämiseen. Liitteessä 1 on esitetty UML-sekvenssikaaviolla raportin tuottamisen vaiheet.

Raportointiosuuden nykyisessä toteutuksessa valuuttakonversio tehdään tietokannassa SQL-kyselyllä. Tietokanta on erikoistunut joukko-operaatioiden tekemiseen tehokkaasti eikä valuuttakursseja tarvitse siirtää verkon ylitse websovellukseen. Jotta prototyypit selviytyvät valuuttakonversiosta mahdollisimman tehokkaasti, käytetään prototyypeissä olio-kätkömuistia. Kätkömuistina käytetään web-palvelimen muistia ajonaikaisen ympäristön kautta. Prototyypit tallentavat kätkömuistiin hakurakenteita, joiden avulla valuuttakonversio voidaan tehdä tehokkaasti, myös raporttien sisältämien tilien hierarkia haetaan kätkömuistiin, jolloin raporttien tuottamisen pitäisi nopeutua. Kätkömuistissa olevaa tietoa on virkistettävä säännöllisin väliajoin, jotta tietokannassa tapahtuneet muutokset (esimerkiksi valuuttakurssien päivitykset) näkyvät raporteissa.

Prototyypit käyttävät ajonaikaisen ympäristön tarjoamaa yhteysallasta kytkeytymiseen tietokantaan, molemmat prototyypit kutsuvat samoja tallennettuja proseduureja, joten tietokantarajapinnan ei pitäisi aiheuttaa vaihteluita suorituskykyyn prototyyppien välillä. Tietokanta pystyy optimoimaan tallennettujen proseduurien peräkkäisiä suorituksia, sillä se osaa hyödyntää jo kertaalleen käännetyn proseduurin suoritussuunnitelmaa.

Prototyypit kehitettiin toiminnallisuudeltaan lähes nykyisen toteutuksen tasolle. Muutamat vähiten käytetyt raporttityypit jätettiin prototyyppien ulkopuolelle, koska niiden toteuttaminen ei olisi tuonut mitään lisäarvoa suorituskykymittauksiin tai vaatinut erityyppisten käyttöliittymäkomponenttien käyttöä. Puuttuvien raporttien lisääminen prototyyppeihin on tehtävissä varsin suoraviivaisesti, sillä ne on huomioitu liiketoimintalogiikkakerroksessa. Käyttäjien autentikointimenetelmä jätettiin prototyypeissä avoimeksi eikä autentikointia käytetä suorituskykymittauksissa. Tietoturvaan kiinnitettiin kuitenkin huomiota ja prototyypit noudattavat samaa turvallisuusmallia kuin nykyinen toteutus.

Prototyypit testattiin laadunvarmistusympäristössä ja niiden tuottamia raportteja verrattiin nykyisen toteutuksen antamiin raportteihin. Pienten korjausten jälkeen prototyypit antoivat oikeanlaisia raportteja ja raporttien luvut täsmäsivät.

2.5 Raportointiosuuden ASP.NET-prototyyppi

2.5.1 Yleistä

ASP.NET-prototyyppi rakennettiin käyttämällä Microsoftin Visual Studio 2003 -kehitysympäristöä. Työkaluja valittaessa tarkistettiin ASP.NET-kehitykseen sopiva Microsoftin ilmainen ASP.NET WebMatrix. ASP.NET WebMatrix osoittautui hyvin yksinkertaiseksi editoriksi, joka ominaisuuksiltaan sopii lähinnä ASP.NET-sivujen ulkoasujen laatimiseen. Web Matrix ei esimerkiksi avusta ollenkaan ohjelmakoodin kirjoittamisessa eikä sisällä virheenjäljitintä (eng. debugger). Prototyypin ohjelmointikieleksi valittiin C#, mikä helpotti logiikka- ja esityskerroksen muuntamista syntaksiltaan samankaltaiselle Java-kielelle JSF-prototyyppiä kehitettäessä. Prototyyppi käyttää ASP.NETin versiota 1.1. Sivujen testaamiseen käytettiin Microsoft Windows 2000 -koneelle asennettua Microsoftin Internet Information Server 5.0 web-palvelinta.

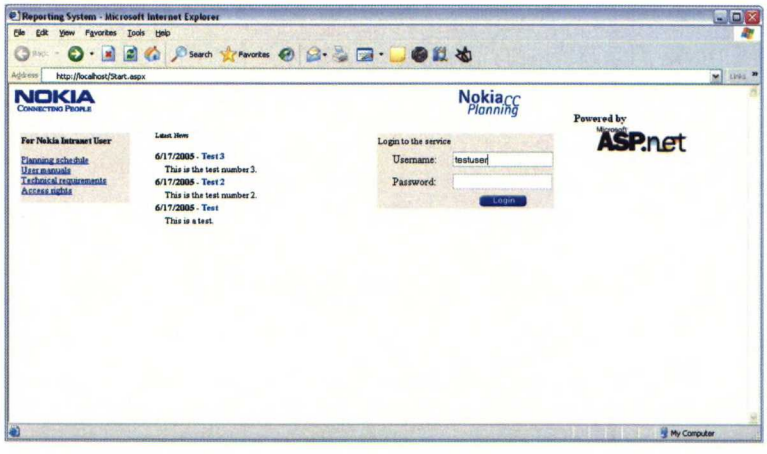
2.5.2 ASP.NET-spesifiset ominaisuudet

Prototyypin datakerroksen suorituskykyä pyrittiin lisäämään käyttämällä ADO.NET-kirjaston tarjoamaa tietokantayhteysallasta. Yhteysaltaan minimi- ja maksimikokoa voidaan säätää prototyypin web.config-nimisestä konfiguraatiotiedostosta. Yhteysaltaan käytön voi konfiguraatiotiedostoa muokkaamalla ottaa pois päältä, jolloin tietokantaan avataan tarvittaessa yhteys jokaisella sivupyynnöllä. Tietokannan kanssa kommunikointiin käytettiin System.Data.SqlClient-nimiavaruudesta löytyviä luokkia, koska nämä tarjoavat paremman suorituskyvyn kytkeydyttäessä Microsoftin SQL Server 2000 -tietokantapalvelimeen kuin yleisemmän tason ODBC-luokat [MVBM04]. Tallennettuja proseduureja kutsuttiin käyttämällä SqlCommand-luokkaa ja tietokannan palauttamia tietueita luettiin SqlDataReader-luokan avulla. SqlDataReader-luokka tukee pelkästään tietojen lukemista ja käyttää vain eteenpäin siirrettävää kursoria. Liiketoimintalogiikkakerrosta pyrittiin optimoimaan tallentamalla usein käytettyä, mutta harvoin muuttuvaa dataa kätkömuistiin. Kätkömuisti toteutettiin omana luokkana, mutta sisäisesti se käyttää ASP.NETin System.Web.Caching-nimiavaruudesta löytyvää Cache-luokkaa. Cache-luokan instanssiin saadaan viittaus käyttämällä HttpContext-luokkaa. Cache-luokka tarjoaa mahdollisuuden oliota lisättäessä määrittää ajan, jonka olio pidetään kätkömuistissa. Kätkömuistiin tallennetut oliot ovat kaikkien asiakkaiden yhteisessä käytössä.

2.5.3 Sivujen toteutus

Prototyypin ASP.NET-sivut luotiin Visual Studio.NET 2003 -kehitysympäristöllä. Sivujen HTML-koodia ei tarvinnut editoida lainkaan käsin. Prototyyppien kaikki sivut pystyttiin toteuttamaan ASP.NETin peruskomponenteilla. Komponentit asemoitiin käyttäen ruudukkoasemointia virta-asemoinnin sijasta. Ruudukko-asemointi on hieman nopeampaa ja lähempänä työpöytäohjelmien käyttöliittymien asemointia. Ruudukkoasemoinnissa komponenttien paikat määritetään absoluuttisina etäisyyksinä sivun ylä- ja alareunasta kun virta-asemoinnissa asemointi tehdään pääsääntöisesti käyttämällä HTML-taulukoita ja CSS-tyylimäärityksiä. Virta-asemointi on parempi vaihtoehto kun sivujen halutaan olevan joustavan kokoisia. Ruudukkoasemointi on yleensä ongelmallisempi eri selaimille.

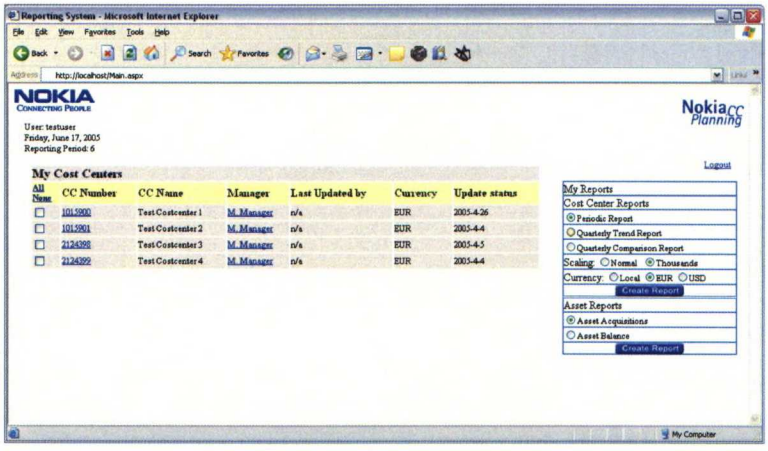
Kuvassa 25 on esitetty prototyypin aloitussivu. Aloitussivu sisältää kolme Label-komponenttia, kaksi TextBox-komponenttia ja yhden Button-komponentin. Kuvat on esitetty käyttämällä Image-komponentteja. Aloitussivulla näkyvät uutiset tuottaa tarkoitusta varten laadittu käyttäjäkomponentti. Käyttäjäkomponentti sisältää Table-komponentin, jonka soluihin lisätään dynaamisesti Label-komponentteja. Käyttäjä-komponentti hyödyntää oliokätkömuistia ja päivittää uutiset tietokannasta tietyin väliajoin, jotta tuoreimmat tiedotukset näkyisivät käyttäjille. Sivun vasemmassa reunassa oleva valikko on toteutettu HTML-taulukkona.



Kuva 25: ASP.NET-protyypin aloitussivu.

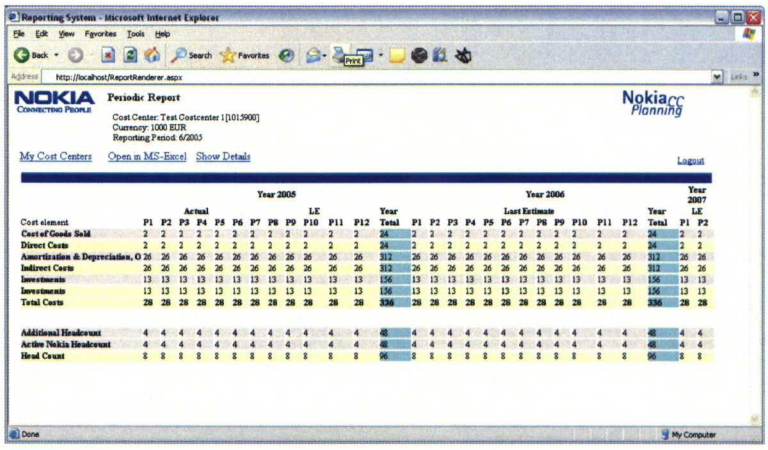
Prototyypin pääsivu on esitetty kuvassa 26. Pääsivu sisältää otsikkokentän, listan kustannuspaikoista ja valikon, josta valitaan raportin tyyppi, kohdevaluutta ja lukujen skaalaus. Valikko on staattinen HTML-taulukko, johon on sijoitettu RadioButton- ja

ImageButton-komponentteja. Kustannuspaikat listataan Table-komponentin tuottamaan dynaamiseen taulukkoon. Taulukon soluihin lisätään dynaamisesti luotuja CheckBox-, LinkButton- ja HyperLink-komponentteja. Otsikkokenttä koostuu kolmesta Label-komponentista. Kuvat on aloitussivun tapaan esitetty Image-komponenteilla.



Kuva 26: ASP.NET-prototyypin pääsivu.

Raporttisivu (esitetty kuvassa 27) koostuu raportin otsikkotiedoista ja itse raportista. Näiden lisäksi sivulla on hyperlinkkejä, joilla voi palata pääsivulle, poistua sovelluksesta tai näyttää raportin yksityiskohtaisemmat rivit. Otsikkotiedot on esitetty Label-komponenteilla, hyperlinkit on toteutettu LinkButton-komponenteilla ja itse raporttiosuus on tehty Table-komponentilla. Raportin renderöinti tapahtuu käymällä Report-olion sisältämät ReportRow-tyyppiset rivit lävitse (kts. Liite 2). Erityyppiset rivit värjätään eri väreillä muuttamalla TableCell-komponenttien taustaväriä. Raportti voi sisältää tavallisia rivejä, yksityiskohtaisia rivejä, välitulosrivejä ja kokonaistulosrivejä. Raportin rivi koostuu ReportItem-olioista, joihin voi liittyä suuntaa-antavia renderöintiohjeita (RenderingDirective-tyyppisiä olioita).



Kuva 27: ASP.NET-prototyypin raporttisivu.

2.6 Raportointiosuuden JavaServer Faces -prototyyppi

2.6.1 Yleistä

JavaServer Faces -prototyyppi rakennettiin sen jälkeen kun ASP.NET-prototyyppi oli valmis. Kehitystyökaluiksi valittiin Borlandin JBuilder 2005 ja Sun Microsystemsin Java Studio Creator. JBuilderillä kirjoitettiin prototyypin data- ja logiikkakerrokset ja sivut laadittiin Java Studio Creatorilla. Kahden työkalun käyttö oli tarpeellista, koska JBuilderin JSF-kehitysominaisuudet eivät ole riittävän graafiset vaan lähinnä avustavat JSF-elementtien kirjoittamisessa. Java Studio Creator tarjosi helppo-käyttöisen graafisen ympäristön JSF-sivujen luomiseen. Java Studio Creatorin ohjelmakoodieditori oli niin hidas, että sitä käytettiin vain esityskerroksen ja logiikkakerroksen integrointiin.

JSF-kehitystyökaluja valittaessa (syksyllä 2004) tarjonta ei ollut kovinkaan runsasta, Oraclen JDeveloper olisi ollut potentiaalinen valinta, mutta Java Studio Creator vakuutti helppokäyttöisyydellään ja integroidulla sovelluspalvelimella. Prototyyppi kirjoitettiin siten, että ASP.NET-prototyyppin data- ja logiikkakerrokset käännettiin luokka luokalta C#:sta Javalle. Java-luokkien paketit nimettiin vastaamaan C#-luokkien nimiavaruuksia. Javasta käytettiin työssä versiota 1.4. Prototyyppiä testattiin kehitysvaiheessa Java Studio Creatorin kanssa toimitetulla sovellus-palvelimella, prototyypin valmistuttua testejä ajettiin BEA WebLogic Server 8.1 -sovelluspalvelimella. JSF-toteutuksena käytettiin Sun Microsystemsin referenssi-toteutusta versiolla 1.1.01.

2.6.2 JSF-spesifiset ominaisuudet

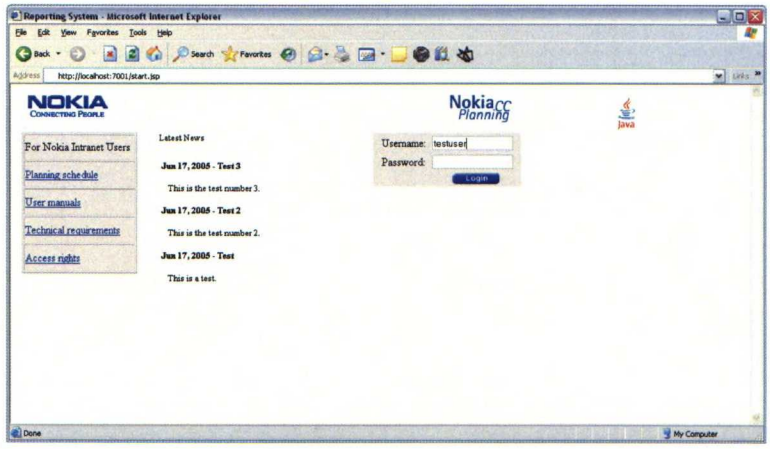
Prototyypin datakerros kytkettiin sovelluspalvelimen tarjoamaan yhteysaltaaseen ja se saa tietokantayhteydet käyttöönsä JNDI-puusta löytyvän java.sql.DataSource-rajapinnan toteuttavan olion kautta. Sovelluspalvelimessa yhteysallas asetettiin käyttämään Microsoftin tarjoamia tyypin neljä JDBC-tietokanta-ajureita. Tyyppi neljä tarkoittaa, että ajurit käyttävät tietokannan kanssa kommunikointiin tietokannan natiiveja protokollia, jolloin suorituskyvyn tulisi olla parempi kuin geneerisiä ajureita käytettäessä [Shi03]. Tallennettuja proseduureja kutsuttiin käyttämällä java.sql. CallableStatement-rajapintaa ja tietokannan palauttamia tietoja luettiin java.sql. ResultSet-rajapinnan kautta. ResultSet-rajapinnan kautta tietoja ei voi muuttaa ja kursoria voi siirtää vain eteenpäin.

Oliokätkömuisti toteutettiin erillisenä luokkana ja se käyttää sisäisesti javax.faces.context-paketista löytyvän ExternalContext-luokan tarjoamaa haku-rakennetta. Hakurakenteeseen on mahdollista tallettaa avaimia ja arvoja pareittain, ja kaikki rakenteeseen tallennetut oliot ovat koko sovelluksen yhteisessä käytössä. Toisin kuin ASP.NET-prototyypissä, arvoihin ei voi liittää vanhenemisaikoja, joten toiminnallisuus piti toteuttaa itse. Prototyypissä päädyttiin ratkaisuun, jossa oliokätkömuistista arvoa haettaessa tarkastetaan onko kyseinen arvo vanhentunut. Mikäli arvo on vanhentunut, kätkömuisti poistaa sen hakurakenteesta ja ilmoittaa pyytäjälle ettei arvoa löytynyt. Ratkaisu ei ole optimaalinen, sillä se pitää muistia varattuna turhaan, tämä ei kuitenkaan ole ongelma, koska prototyyppi tallentaa kätkömuistiin vain sellaisia arvoja, joita kysytään toistuvasti. Ratkaisu ei myöskään vaadi hakurakenteen ajoittaista lukitsemista ja läpikäyntiä.

2.6.3 Sivujen toteutus

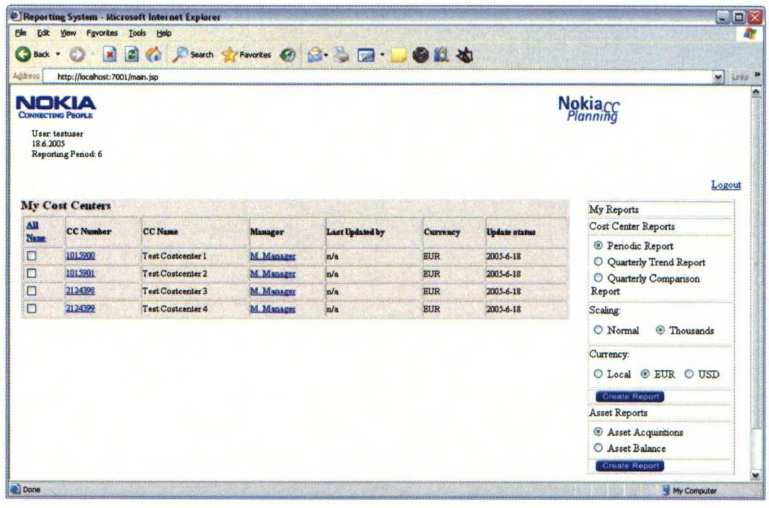
JSF-prototyypin sivut laadittiin kokonaan Java Studio Creatorilla. Sivujen luonti ei vaatinut lainkaan manuaalista HTML-koodin muokkaamista. Komponentit aseteltiin käyttäen ruudukkoasettelua. Prototyyppi pystyttiin rakentamaan käyttämällä JSF:n HTML-kieltä tuottavia peruskomponentteja, jotka löytyvät paketista javax.faces.component.html.

JSF-prototyypin aloitussivu on esitetty kuvassa 28. Sivulla olevat kuvat ovat HtmlGraphicImage-komponentteja. Komponenttien ryhmittely toteutettiin HtmlPanelGrid-komponenteilla. Käyttäjätunnus syötetään HtmlInputText-komponenttiin ja salasana HtmlInputSecret-komponenttiin. Login-nappi on toteutettu HtmlCommandButton-komponentilla ja sille on määritelty kuva sen image-attribuutilla. Uutiset esitetään HtmlPanelGrid-komponentissa, johon lisätään dynaamisesti HtmlOutputText-komponentteja, listaa viimeisimmistä uutisista pidetään oliokätkömuistissa ja se päivitetään tietyin väliajoin.



Kuva 28: JSF-prototyypin aloitussivu.

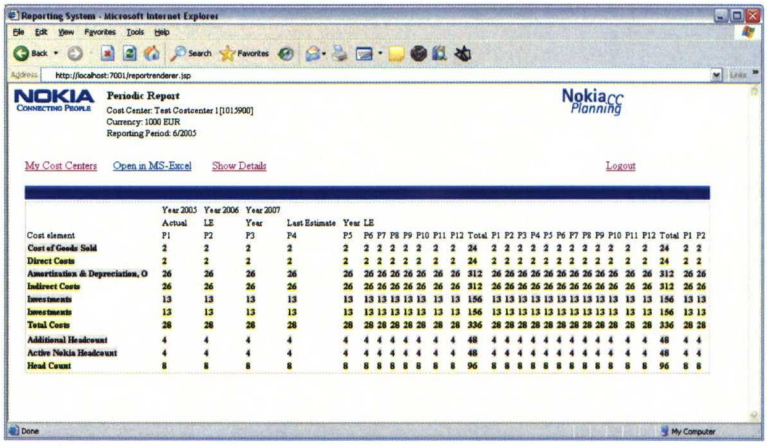
Prototyypin pääsivu (kuvassa 29) sisältää listan kustannuspaikoista ja valikon, molemmat elementit toteutettiin HtmlGridPanel-komponenteilla, joihin upotettiin lapsikomponentteja. Raportin tyyppi, skaalaus ja kohdevaluutta valitaan HtmlSelectOneRadio-komponenteilla. Kustannuspaikkalista luodaan dynaamisesti, lista sisältää HtmlSelectBooleanCheckBox-, HtmlCommandLink-, HtmlOutputLink- ja HtmlOutputText-komponentteja.



Kuva 29: JSF-prototyypin pääsivu.

Prototyypin raporttisivu on esitetty kuvassa 30. Raporttisivu sisältää raportin otsikkotiedot ja varsinaisen raporttiosuuden. Molemmat osuudet esitetään käyttämällä HtmlGridPanel-komponentteja. Raporttiosuuden sarakkeiden lukumäärä määräytyy dynaamisesti raportin sisältämien alkioiden mukaisesti. Raporttitaulukon solut ovat HtmlOutputText-komponentteja ja niiden ulkoasua muokataan tyyli-

määrittelyjen kautta. Raporttitaulukon alussa näytetään vuodet ja kuukaudet. Yhden vuosi-solun tulisi leveydessä kattaa sen sisältämien kuukausien sarakkeet, mutta HtmlGridPanel-komponentti ei tukenut tällaista ominaisuutta vaikka se renderöi itsensä HTML-taulukoksi. Sivulla näkyvät hyperlinkit toteutettiin HtmlCommand-Link-komponenteilla.



Kuva 30: JSF-prototyypin raporttisivu.

2.7 Yhteenveto

Tässä luvussa esiteltiin tuotantokäytössä oleva KP-järjestelmä ja sen raportointiosuus. KP-järjestelmä on tarkoitettu kustannuspaikkojen menojen suunnitteluun ja seurantaan. Järjestelmän raportointiosuus on teknisesti toteutettu JSP-sivuina ja toteutukseen liittyy ylläpitoa vaikeuttavia ongelmia, jotka tämän työn yhteydessä pyrittiin ratkaisemaan käyttöliittymäkomponenttipohjaisilla prototyypeillä. Prototyypit toteutettiin ASP.NET- ja JavaServer Faces -komponenttiteknologioilla. Prototyyppien sivujen tuli olla ulkoasultaan mahdollisimman samankaltaisia JSP-toteutuksen kanssa. Suorituskyvyltään prototyyppien piti olla vähintään yhtä hyviä kuin JSP-toteutus.

Prototyypit toteutettiin noudattamalla kolmikerrosarkkitehtuuria, jossa sovellus jaettiin esitys-, liiketoimintalogiikka-, ja datakerrokseen. Kukin kerros mallinnettiin UML-luokkakaavioilla ja tärkeimmät toiminnot UML-sekvenssikaavioilla. Prototyypit ovat liiketoimintalogiikka- ja datakerrokseltaan mahdollisimman yhtenevät ja käyttävät samaa koodirunkoa. JSF-prototyyppi kirjoitettiin Javalla ja ASP.NET-prototyyppi C#:lla. Esityskerros on toteutettu käytetyn komponenttiteknologian ehdoilla, sivuston rakenne on kuitenkin molemmissa prototyypeissä sama. Prototyyppien suorituskykyä pyrittiin parantamaan varastoimalla usein käytettyä mutta harvoin muuttuvaa tietoa oliokätkömuistiin.

Molempien prototyyppien sivut pystyttiin toteuttamaan komponenttiteknologioiden tarjoamien peruskomponenttien avulla. Sivujen rakentamisessa hyödynnettiin graafisia kehitysympäristöjä, ASP.NET-prototyypin osalta Microsoftin Visual Studio 2003:sta ja JSF-prototyypin sivut rakennettiin Sun Microsystemsin Java Studio Creatorilla.

3 Toteutuksien vertailu

3.1 Johdanto

Tämän luvun tarkoituksena on verrata keskenään ASP.NET- ja JavaServer Faces -teknologioita sekä työn ensimmäisessä luvussa esiteltyjen arkkitehtuurikuvausten perusteella että prototyyppien rakentamisessa saatujen käytännön kokemusten pohjalta. Vertailua suoritetaan myös JSP-toteutusta vasten niissä kohdin kuin sen katsotaan olevan mielekästä. Vertailu on jaettu kolmeen osa-alueeseen. Aluksi vertaillaan kehittäjän näkökulmasta, jolloin arvioidaan kehitystyön tuottavuutta, kompleksisuutta ja tukea erilaisille selaimille. Toisessa osassa vertailu suoritetaan tuotantovaiheen näkökulmasta ja tarkoituksena on selvittää pärjäävätkö komponenttiteknologioilla rakennetut prototyypit tuotannossa olevalle JSP-toteutukselle. Luvun loppupuolella vertailu tehdään ylläpidon näkökulmasta. Arvioitavana on komponenttiteknologioiden ylläpidettävyys verrattuna toisen sukupolven web-teknologioihin ja kuinka uudelleenkäytettäviä käyttöliittymäkomponentit ovat.

3.2 Kehitysprosessi

3.2.1 Tuottavuus

Tuottavuudella tarkoitetaan tässä yhteydessä kykyä rakentaa nopeasti monipuolinen web-käyttöliittymä, jonka taakse on sijoitettu sitä ohjaavaa logiikka. Sekä ASP.NET että JSF nopeuttavat web-käyttöliittymien rakentamista merkittävästi, koska käyttöliittymien rakentaminen on mahdollista ilman HTML-kielen tuntemista. Kehittäjän täytyy vain hallita käyttöliittymäkomponenttien käyttö ja HTML-kielen tuottaminen jää järjestelmän vastuulle. Työssä toteutettujen prototyyppien web-käyttöliittymien rakentamiseen kului aikaa vain joitakin päiviä kun JSP-toteutuksen sivuja rakennettiin aikoinaan kaksi viikkoa. Käyttöliittymää ohjaavan logiikan sijoittaminen itse sivun määrityksen ulkopuolelle helpotti kehitystyötä, koska samassa lähdetiedostossa ei ollut sekaisin ohjelmointikieltä ja kuvauskieltä. Tapahtumien käsittelijöiden sitominen käyttöliittymäkomponentteihin onnistui vaivatta molemmissa prototyypeissä.

Microsoftin Visual Studio.NET -kehitysympäristöä käytettäessä ASP.NET-sivujen kehitystyö ei eroa juuri lainkaan työpöytäsovellusten käyttöliittymien rakentamisesta.

ASP.NET ja JSF nostavat web-käyttöliittymien rakentamisen yhden abstraktiokerroksen verran ylemmäksi verrattuna toisen sukupolven dokumenttipohjaisiin webteknologioihin. Kehittäjän ei enää tarvitse työskennellä HTML-elementtien ja HTTP-pyyntöjen parissa vaan ohjelmointi tapahtuu käyttöliittymäkomponentteja ja niiden aiheuttamien tapahtumien tasolla. Mikäli kehitystyö halutaan jakaa osiin siten, että ohjelmoijat kirjoittavat logiikan ja web-suunnittelija tekee sivut, auttavat käyttöliittymäkomponentit tässäkin. Sekä JSF- että ASP.NET-kehityksen tuottavuuteen vaikuttaa suuresti käytetyn kehitysympäristön ominaisuudet. Pelkkää tekstieditoria käytettäessä kehitystyön tuottavuus putoaa alle JSP- tai ASP-kehityksen tuottavuuden, koska käyttöliittymäkomponenttien kuvauskieliset määritykset ovat huomattavasti JSP- ja ASP-määrityksiä monimutkaisempia. Käyttöliittymäkomponenttien asettelu web-sivulle onnistui ongelmitta sekä Visual Studio.NETillä että Java Studio Creatorilla eikä kehitysympäristöjen käytettävyydessä ollut huomauttamista.

3.2.2 Teknologian kypsyys

Kummankaan prototyypin rakentamisessa ei törmätty ylitsepääsemättömiin ongelmiin. ASP.NETin peruskomponenttitarjonta on JSF:ää monipuolisempi ja erityisesti ASP.NETin Table-komponentti tarjosi hyvät mahdollisuudet dynaamisen taulukkomuotoisen tiedon esittämiseen. JSF-prototyypissä taulukkomuotoisen tiedon esittämiseen kokeiltiin aluksi HtmlDataTable-komponenttia, mutta se vaatii sidonnan dynaamisen tietolähteeseen kuten tietokannasta haettuun tauluun tai listaan [Man04]. Prototyypissä päädyttiin vaihtoehtoisten komponenttien puutteen takia käyttämään taulukkomuotoisen tiedon esittämiseen HtmlGridPanel-komponenttia, vaikka siitä puuttui ominaisuuksia kuten solutasoinen ulkoasun määritys ja useamman sarakkeen levyiset solut.

ASP.NET on ollut markkinoilla huomattavasti pidempään kuin JSF ja kolmansien osapuolten komponenttitarjonta onkin paljon suurempi ASP.NETille. Microsoftin Visual Studio.NET -kehitysympäristön ylivalta-asema ASP.NET-kehityksessä on kiistaton, mikä yksinkertaistaa komponenttien rakentamista ja testaamista. JSF-kehitysympäristöjä on suuri määrä ja komponenttien toiminta eri kehitysympäristöissä on vaihtelevaa. Ongelma on tiedostettu ja komponenttien kehitysaikaista toimintaa ja paketointia pyritään standardoimaan. Oracle on jättänyt kesäkuussa 2005 Java-spesifikaatiopyynnön [JSR-276], jonka tarkoituksena on

muodostaa yhtenäinen metadata-malli JSF-komponenteille. Ongelmaan saadaan ratkaisu varmasti lähitulevaisuudessa, mutta tällä hetkellä kolmansien osapuolten valmistamien komponenttien käyttäminen web-kehityksessä onnistuu helpommin ASP.NETillä.

3.2.3 Kompleksisuus

Prototyyppien käyttöliittymien rakentaminen ASP.NETillä ja JSF:llä oli verraten helppoa kun käytettiin graafisia kehitysympäristöjä. Kehitysympäristöjen suunnittelunäkymä vastasi hyvin pitkälle sitä millaisena sivu näkyi selaimessa. Käyttöliittymäkomponentit aseteltiin kehitysnäkymässä vedä-ja-pudota -periaatteella ja kehitysympäristö muokkasi sivun kuvauskielistä määritystä automaattisesti. Komponenttien ominaisuuksia pystyi muuttamaan kehitysnäkymän tarjoaman editorin avulla ja ulkoasumuutokset näkyivät heti. JSF:n sivumääritykset ovat monimutkaisempia kuin ASP.NETin. Tämä johtuu siitä, että ASP.NET-komponenttien kuvauskieliset määritykset eivät sisällä viittauksia ohjelmakoodia sisältävään taustatiedostoon. JSF-komponenttien määritykset viittaavat avustaviin papuihin ja määrityksissä käytetään EL-kieltä. JSF tarjoaa kuitenkin joustavamman mallin, koska yksi JSF-sivu voi hyödyntää useaa avustavaa papua, kun ASP.NET-sivu on sidottu yhteen taustatiedostoon. ASP.NET-taustatiedostossa on kuvattu sivun koko komponenttipuu ja sen rakentaminen, kun JSF-komponenteille ei avustavassa pavussa tarvitse välttämättä määrittää ilmentymää vaan komponentin arvo voidaan sitoa pavun julkiseen ominaisuuteen. Koska JSF mahdollistaa erilaisten renderöinti-kirjastojen käytön, niin sen komponentit on määritelty yleisemmällä tasolla kuin ASP.NETissä. Jopa JSF:n HTML-spesifiset komponentit ovat geneerisempiä kuin ASP.NET-komponentit. Esimerkiksi ASP.NETin TextBox-komponentin tekstiarvo asetetaan ja luetaan sen String-tyyppisen ominaisuuden kautta. JSF:n vastaava HTML-spesifinen komponentti on HtmlInputText-komponentti ja sen arvo asetetaan ja luetetaan Object-tyyppisen arvoa käyttävien metodien kautta. ASP.NETin TextBox-komponentin ulkoasua voidaan muokata sen julkisten ominaisuuksien kuten taustavärin, fontin, reunan värin ja reunan paksuuden kautta, kun JSF HtmlInputText-komponentin ulkoasua voi muokata vain antamalla sille CSS-tyylimäärityksen merkkijonona. Ero ei ole suuri, mutta ASP.NET-komponentit noudattavat tässä mielessä paremmin komponenttiajattelua. Käyttöliittymäkomponenttien dynaamisten lisääminen sivulle onnistuu yhtä helposti

sekä ASP.NETillä että JSF:llä. Dynaaminen ASP.NET-komponentti tehdään luomalla komponentista ilmentymä ja lisäämällä se komponenttipuuhun. Dynaamiset JSF-komponentit luodaan Application-oliolla. Dynaamisten komponenttien sitominen tapahtumankäsittelijöihin onnistuu ASP.NETissä hieman helpommin, koska sidonta onnistuu delegaateilla, jotka ovat tyyppiturvallisia funktio-osoittimia [RCGHMMNSW01]. Kuvassa 31 on esitetty C#-kielellä LinkButton-komponentin luonti dynaamisesti ja sen Click-tapahtuman sidonta DisplayCostCenterInfo-nimiseen tapahtumankäsittelyfunktioon.

```
LinkButton linkbCostCenter = null;
linkbCostCenter = new LinkButton();
linkbCostCenter.Click +=new EventHandler(this.DisplayCostCenterInfo);
```

Kuva 31: LinkButton-komponentin luonti ja sitominen tapahtumankäsittelijään.

JSF-komponentin sitominen tapahtumankäsittelijään tehdään luomalla ensin Java-kielellä MethodBinding-olio, jonka alustustaminen vaatii EL-kielisen merkkijonon antamista parametrina. Kuvassa 32 on esitetty dynaamisen HtmlCommandLink-komponentin luonti ja sen sitominen Main-nimisen avustavan pavun recordCC-nimiseen tapahtumankäsittelijämetodiin. JSF-komponentin sitominen ei siis onnistu pelkästään Javalla vaan vaatii myös EL-kielen tuntemista.

```
HtmlCommandLink linkCC = null;
linkCC = (HtmlCommandLink) app.createComponent(
            javax.faces.component.html.HtmlCommandLink.COMPONENT_TYPE);
linkCC.setActionListener(app.createMethodBinding("#{Main.recordCC}",
            new Class[] {javax.faces.event.ActionEvent.class }));
```

Kuva 32: HtmlCommandLink-komponentin luonti ja sitominen tapahtumankäsittelijään.

3.2.4 Selainriippumattomuus

Prototyyppejä testattiin Internet Explorer 5, Internet Explorer 6, Mozilla Firefox 1.0.4, Opera 8.01, Netscape 4.75 ja Netscape 8.0.2 selaimilla. Molempien prototyyppien sivut toimivat moitteetta kaikissa testatuissa selaimissa paitsi Netscape 4.75:ssä. Netscape 4.75 selaimella komponenttien paikat eivät olleet oikeat ja muitakin piirtohäiriöitä ilmeni, selain ei ymmärtänyt komponenttien absoluuttisia paikanmäärityksiä. Sivut eivät kuitenkaan aiheuttaneet selaimessa virheilmoituksia. Toimimattomuuden Netscape 4.75:ssä ei katsottu kuitenkaan olevan merkittävä

ongelma ainakaan KP-järjestelmän kannalta, sillä vanhat selaimet eivät ole suosittuja yrityskäytössä erityisesti tietoturvariskien takia. JSF-implementaation mukana tuleva standardi HTML-renderöintikirjasto on HTML 4.0 -yhteensopiva [Man04]. Vaihtamalla renderöintikirjastoa JSF-sivut on teoriassa mahdollista saada tukemaan vanhempiakin selaimia. Java Studio Creatorilla luotujen JSF-sivujen dokumenttityyppi oli XHTML 1.0 Transitional.

ASP.NET jakaa selaimet kahteen eri luokkaan, joista alempaan kuuluvat vanhat HTML 3.0 -versiota ymmärtävät selaimet ja ylempään luokkaan uudet HTML 4.0 -versiota tukevat selaimet. Selaimen luokka vaikuttaa siihen, miten ASP.NETin perus-komponentit renderöivät itsensä HTML-kielelle. ASP.NET tunnistaa selaimen HTTP-pyynnössä välitetyn selaintunnisteen avulla, tunnistusta pystyy muuttamaan muokkaamalla ASP.NETin konfiguraatiotiedostoja. ASP.NET-prototyypin tunnistusta muokattiin, jotta se tunnistaisi Firefox ja Netscape selaimet ylempään ryhmään kuuluviksi. Prototyyppejä testattiin myös niin, että selaimen JavaScript-tuki oli kytkettynä pois. Prototyypit eivät toimineet kunnolla, koska sekä ASP.NET että JSF hyödyntävät JavaScriptiä HTML-lomakkeiden lähettämisessä. ASP.NET tukee JavaScriptin lisäksi VBScript-kieltä asiakaspään skriptikielenä, mutta se ei toimi muissa kuin Microsoftin selaimissa.

3.2.5 Tuki mobiililaitteille

Sekä ASP.NET että JSF tukevat mobiililaitteiden käyttöä. Mobiilaitteella tarkoitetaan tässä yhteydessä kämmentietokonetta, älypuhelinta tai niihin verrattavissa olevaa resursseiltaan rajoittunutta laitetta. JSF sopii arkkitehtuuriltaan paremmin mobiilikehitykseen, koska JSF-sovellus voi sivupyynnön saatuaan valita dynaamisesti laitteelle sopivan renderöintikirjaston ja renderöidä käyttöliittymäkomponentit laitteen ymmärtämään muotoon. JSF ei automaattisesti valitse oikeata renderöintikirjastoa, vaan tuen rakentaminen on sovelluksen kehittäjän vastuulla. Suurin etu on se, että sama sovellus pystyy palvelemaan erityyppisiä päätelaitteita ilman että ohjaimeen tai malliin pitää tehdä muutoksia. ASP.NET tukee mobiililaitteita erityisten mobiililaitteille tarkoitettujen käyttöliittymäkomponenttien kautta (eng. ASP.NET Mobile Controls). Mobiililaitteille on rakennettava oma ASP.NET-sivusto käyttämällä mobiilikomponentteja. Kun mobiililaite lähettää sivupyynnön, niin ASP.NET-ympäristö pyrkii tunnistamaan laitteen ja valitsemaan sille sopivan esitysmuodon. Mikäli laitetta ei tunnisteta oikein, niin se voi saada

vastauksen väärässä formaatissa. Laitteiden ominaisuudet määritetään konfigurointi-tiedostoon, tiedostoa on päivitettävä säännöllisesti, jotta uusien laitteiden tunnistaminen onnistuisi. Microsoft on lopettanut päivitystiedostojen tuottamisen ja mobiilikomponenttien toimintaa tullaan muuttamaan ASP.NETin seuraavassa versiossa [RFMWD-W].

3.3 Tuotantovaihe

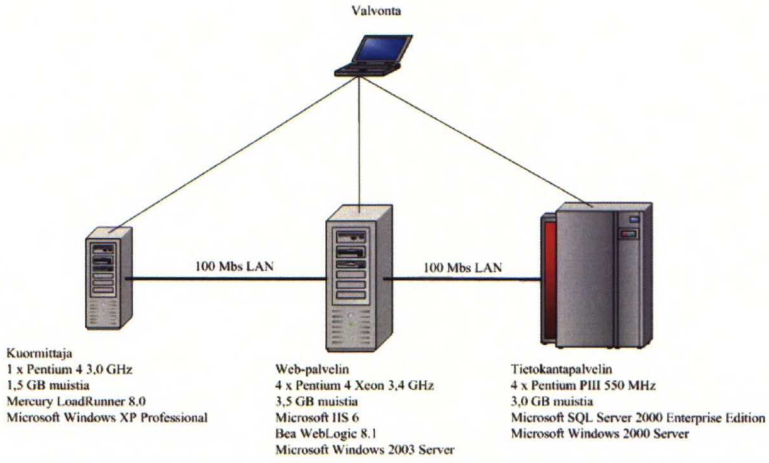
3.3.1 Suorituskykymittaukset

3.3.1.1 Yleistä

Suorituskykymittauksien tarkoituksena oli selvittää kuinka tehokkaita ASP.NET- ja JSF-komponenttiteknologiat ovat verrattuna toisiinsa ja JSP-teknologiaan. Mittaukset kohdistettiin tämän työn yhteydessä rakennettuihin prototyyppeihin ja KP-järjestelmän raportointiosuuden JSP-toteutukseen. Tärkeimmät käytetyt mittarit olivat istuntojen lukumäärä aikayksikköä kohden, istuntojen ja sivupyyntöjen vasteajat ja siirtokyky.

3.3.1.2 Testiympäristö ja mittausmenetelmät

Testiympäristö käsitti neljä eri tietokonetta kuten kuvassa 33 on esitetty. Yhtä koneista käytettiin testien valvontaan, toinen kone kuormitti web-palvelinta ja suoritti mittaukset. Web-palvelin kommunikoi testien aikana tietokantapalvelimen kanssa.



Kuva 33: Testiympäristön kokoonpano.

Kuormittava kone ajoi Microsoft Windows XP Professional - käyttöjärjestelmää ja sisälsi yhden Intel Pentium 4 3,0 GHz prosessorin ja 1,5 GB käyttömuistia. Web-palvelimen käyttöjärjestelmä oli Microsoft Windows 2003 Server, palvelin sisälsi neljä Intel Pentium 4 Xeon 3,4 GHz suoritinta ja 3,5 GB käyttömuistia. Tietokantapalvelin sisälsi neljä Intel PIII 550MHz prosessoria ja käyttömuistia oli 3,0 GB. Tietokantapalvelimen käyttöjärjestelmä oli Microsoft Windows 2000 Advanced Server ja tietokantaohjelmistona Microsoftin SQL Server 2000 Enterprise Edition. Kaikki koneet sijaitsivat samassa 100 Mbs Ethernet-lähiverkossa.

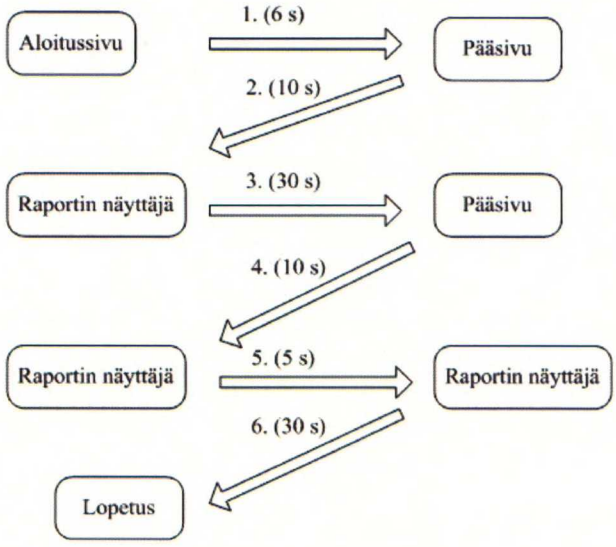
Web-palvelimelle asennettiin Microsoftin IIS 6 web-palvelinohjelmisto, jolla ASP.NET-prototyyppiä ajettiin. JSP-toteutusta ja JSF-prototyyppiä ajettiin Bea WebLogic Server 8.1 SP4 Java-sovelluspalvelimella. Suorituskykymittaukset tehtiin Mercury Interactiven LoadRunner-ohjelmistolla, joka pystyy kuormittamaan web-palvelinta luomalla virtuaalisia käyttäjiä. LoadRunner tallentaa samalla mittaus-tuloksia kun se kuormittaa kohdepalvelinta. Testiskriptejä muokattiin niin, että LoadRunner merkitsee mielenkiintoisimmat sivulataukset transaktioiksi, jolloin se määrittää niille vasteajat ja esiintymistiheydet. Mittauksia varten perustettiin 1900 käyttäjää ja kullakin käyttäjällä oli oikeudet kolmeen eri kustannuspaikkaan. Käyttöoikeudet jaettiin siten, että ne kattoivat 1900 eri KP-järjestelmän kustannuspaikkaa. Mittaukset suoritettiin peräkkäin samana päivänä, testilaitteistoilla ei ollut testien aikana muita käyttäjiä. Kun ASP.NET-prototyyppiä testattiin oli Bea WebLogic Server suljettuna ja vastaavasti JSF-prototyyppiä ja JSP-toteutusta testattaessa IIS 6 oli sammutettuna. Testien tulokset varmistettiin ajamalla ne uudelleen satunnaisessa järjestyksessä.

3.3.1.3 Käytetyt asetukset ja konfiguraatiot

ASP.NET-prototyyppi asennetttiin Microsoftin IIS 6 web-palvelimelle käyttämällä Visual Studio.NETillä tuotettua asennuspakettia. Prototyyppiä varten palvelimelle oli asennettu Microsoftin .NET-sovelluskehyksen versio 1.1.4322. ASP.NET-prototyyppiä ajettiin web-palvelimen natiivissa moodissa, jolloin machine.config-konfiguraatiotiedoston processModel-kohdan asetukset eivät olleet voimassa. Konfiguraatiotiedostoon tehtiin Microsoftin suosittelemat muutokset [MVBM04], jolloin kohtiin minFreeThreads, minLocalRequestFreeThreads ja maxconnection asetettiin web-palvelimen prosessorien lukumäärän perusteella arvot 352, 304 ja 48. Prototyypin web.config-tiedostoa muokkaamalla otettiin testaus- ja virhe-jäljitystoiminteet pois päältä suorituskyvyn maksimoimiseksi. Microsoftin ohjeiden mukaisesti suorituskykyä pyrittiin parantamaan vielä asettamalla raportti-sivulle vain lukuoikeudet istunto-olioon. Prototyyppi asetettiin käyttämään tietokantayhteysallasta siten, että altaan koko on vähintään 100 ja maksimissaan 200.
JSF-prototyyppi asennettiin WAR-tiedostona WebLogic-palvelimelle. Prototyyppi määriteltiin käyttämään sovelluspalvelimen hallinnoimaa tietokantayhteysallasta, jonka vähimmäiskooksi asetettiin 100 ja maksimikooksi 200. Sovelluspalvelin asetettiin kasvattamaan tai pienentämään allasta 10 yhteyden askelin. JSF-prototyyppi asetettiin tallentamaan komponenttipuut palvelimelle selaimen sijasta ja Java-virtuaalikoneeksi määritettiin suositusten mukaisesti [NPBMMNP03] Bean valmistama JRockit JVM.
JSP-toteutus kopioitiin sellaisenaan tuotantoympäristöstä ja siitä poistettiin käyttäjien autentikointi, koska ASP.NET- ja JSF-prototyypit eivät autentikoi käyttäjiä. Lisäksi toteutukseen lisättiin yksinkertainen JSP-sivu, johon käyttäjät ohjataan kun he poistuvat järjestelmästä. Muutos tehtiin, koska toteutus muuten lataisi aloitussivun poistuttaessa, mikä olisi vaikuttanut negatiivisesti toteutuksen suorituskykyyn verrattuna prototyyppeihin. JSP-toteutus asetettiin käyttämään samaa tietokanta-yhteysallasta kuin JSF-prototyyppi. Toteutus asennettiin WAR-tiedostona WebLogic -palvelimelle ja se käyttää samaa Java-virtuaalikonetta kuin JSF-prototyyppi.

3.3.1.4 Testiskenaariot ja suoritetut testit

Mittauksilla haluttiin testata teknologioiden suorituskykyä mahdollisimman todenmukaisissa käyttötilanteissa. Tätä varten mittauksia ei rajoitettu käsittämään vain esimerkiksi raportin luomista, vaan mittauksia varten laadittiin testiskenaario, joka suoritettiin kahdella eri tavalla. Testiskenaariossa pyrittiin simuloimaan loppukäyttäjän toimintaa mahdollisimman tarkasti. Kuvassa 34 on esitetty testeissä käytetty kuusivaiheinen testiskenaario. Skenaario kuvaa tyypillistä loppukäyttäjän istuntoa. Aluksi käyttäjä saapuu sovelluksen aloitussivulle ja syöttää järjestelmälle käyttäjätunnuksen. Käyttäjä ohjataan pääsivulle, jossa näytetään ne kustannuspaikat, joihin käyttäjällä on oikeus. Käyttäjä valitsee kolme kustannuspaikkaa ja pyytää raportin. Kun raportti on luotu, niin se esitetään käyttäjälle. Tämän jälkeen käyttäjä palaa takaisin pääsivulle ja valitsee eri raporttityypin ja kohdevaluutaksi Yhdysvaltojen dollarin. Kun raportti on luotu ja näytetty käyttäjälle, niin käyttäjä haluaa katsoa raportin yksityiskohtaiset rivit, jolloin raportti näytetään uudelleen muotoiltuna. Tämän jälkeen käyttäjä poistuu järjestelmästä ja käyttäjän istunto lopetetaan.



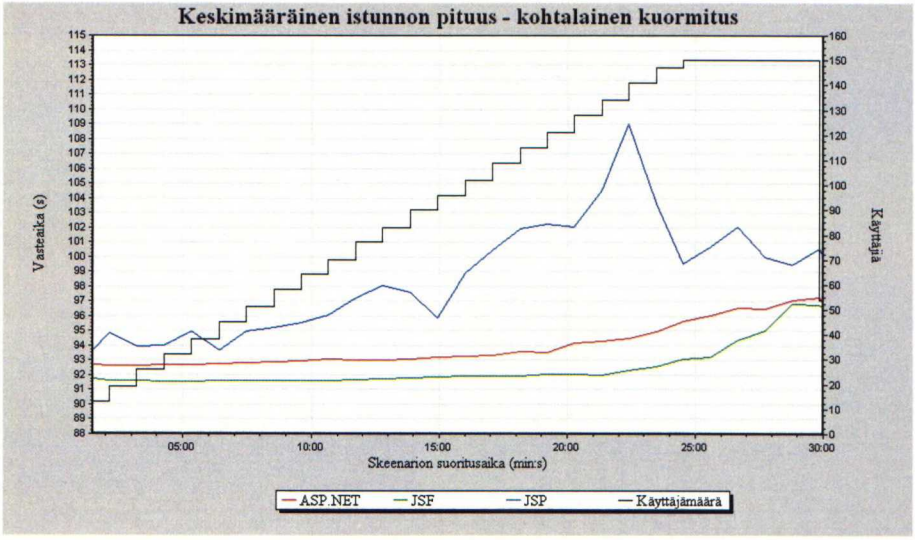
Kuva 34: Testiskeenarion vaiheet.

Testiskenaariossa on huomioitu käyttäjän harkitsemiseen kuluva aika. Käyttäjä ei siirry sivulta toiselle välittömästi vaan käyttää tietyn verran aikaa sivun lukemiseen ja valintojen tekemiseen. Testiskenaariossa käytetään kiinteitä viiveitä sivusiirtymien välillä. Viiveet on merkitty suluissa sekunteina kuvassa 34. Kullekin kolmelle

teknologialle ajettiin kaksi testiä, jotka sisälsivät saman testiskenaarion. Ensimmäinen testi kesti noin 30 minuuttia ja käyttäjämäärä nousi tasaisesti yhdestä aktiivisesta käyttäjästä 150 käyttäjään, jonka jälkeen kuormitusta pidettiin yllä viisi minuuttia. Testin tarkoituksena oli simuloida mahdollisimman todellista käyttäjien aiheuttamaa normaalia kuormitusta. Toinen testi tehtiin kuormittamalla järjestelmiä lineaarisesti kasvavalla käyttäjämäärällä ilman sivusiirtymien välisiä viiveitä. Käyttäjämäärä nostettiin yhdestä käyttäjästä 50 käyttäjään seitsemän minuutin aikana, tämän jälkeen kuormaa pidettiin yllä viisi minuuttia. Testin tarkoituksena oli mitata järjestelmien suorituskykyä hyvin suuren kuormituksen alla.

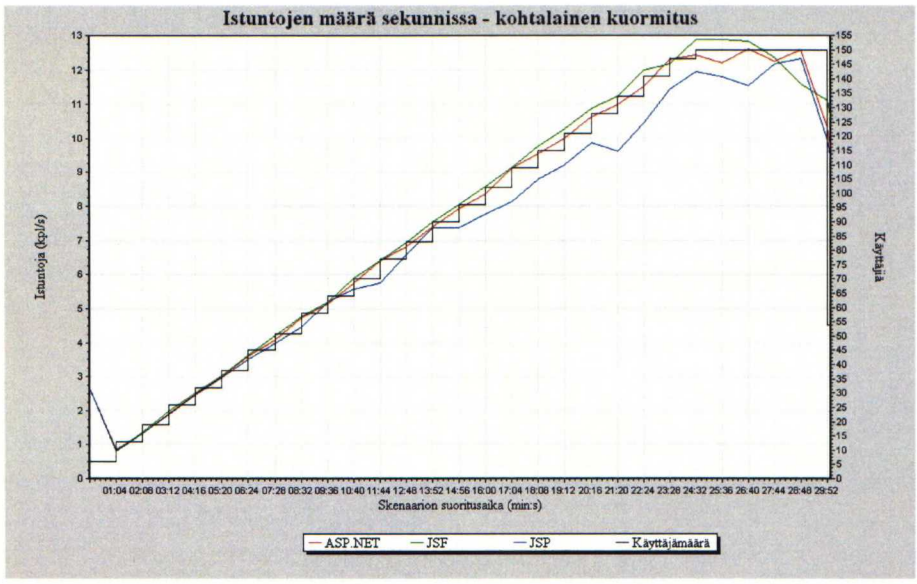
3.3.1.5 Mittaustulokset ja tulosten analysointi

Kuvassa 35 on esitetty keskimääräinen istunnon pituus skenaarion suoritusajan funktiona kun järjestelmiä kuormitettiin kohtalaisella kuormalla. ASP.NET-prototyypin kuvaaja on merkitty punaisella, JSF-prototyypin vihreällä ja JSP-toteutuksen sinisellä värillä. Samassa kuvassa on esitetty myös virtuaalikäyttäjien määrän kuvaaja mustalla värillä. Samaa värityskäytäntöä on sovellettu kaikissa mittaustuloksia esittävissä kuvissa. Istunnon pituudella tarkoitetaan sitä aikaa, joka kuluu yhdeltä virtuaalikäyttäjältä testiskenaarion suorittamiseen. Kohtalaisella kuormituksella tehdyissä mittauksissa käyttäjien harkintaa simuloivat viiveet olivat päällä, joten teoreettinen alaraja istunnon pituudelle on 91 sekuntia. Kuvasta nähdään, että prototyypit suoriutuivat kuormasta paremmin kuin JSP-toteutus. JSF-prototyyppi oli hieman ASP.NET-prototyyppiä nopeampi vaikkakin hidastui kun kuormituksessa seurasi tasainen vaihe.



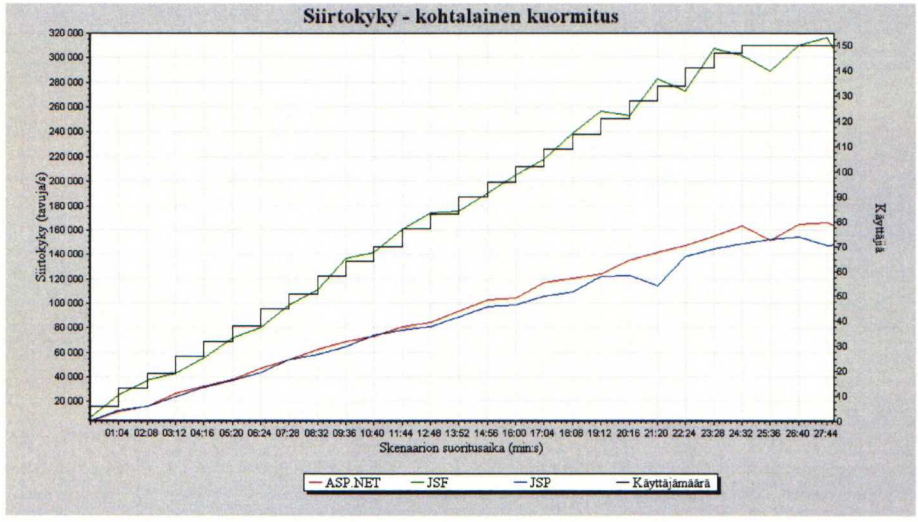
Kuva 35: Keskimääräinen istunnon pituus kohtalaisella kuormituksella.

Kaikki toteutukset selvisivät kohtalaisesta kuormasta moitteetta ja istuntojen määrä kasvoi suorassa suhteessa virtuaalikäyttäjien määrään, kuten kuvasta 36 voidaan todeta. ASP.NET-prototyypin kuvaaja oskilloi vähiten ja JSP-toteutuksen eniten.



Kuva 36: Suorituskyky kohtalaisella kuormituksella.

Mittausten aikana tarkkailtiin kaikkien testilaitteiden prosessorien kuormitusta eikä kohtalaisten kuormitustestien aikana yhdestäkään koneesta loppunut suoritintehot kesken. Tätä väitettä tukee kuva 37, josta nähdään, että siirtokyky (eng. throughput) kasvaa lineaarisesti kunnes virtuaalikäyttäjien määrä ei enää kasvateta. Kuvasta nähdään, että JSF-prototyypin siirtokyky oli huomattavasti suurempi kuin kahden muun toteutuksen, ero ei selity pelkästään paremmalla suorituskyvyllä vaan sillä, että JSF-sivut olivat renderöityinä fyysisesti suurempia ja niiden siirtämiseen kului täten enemmän kaistaa.



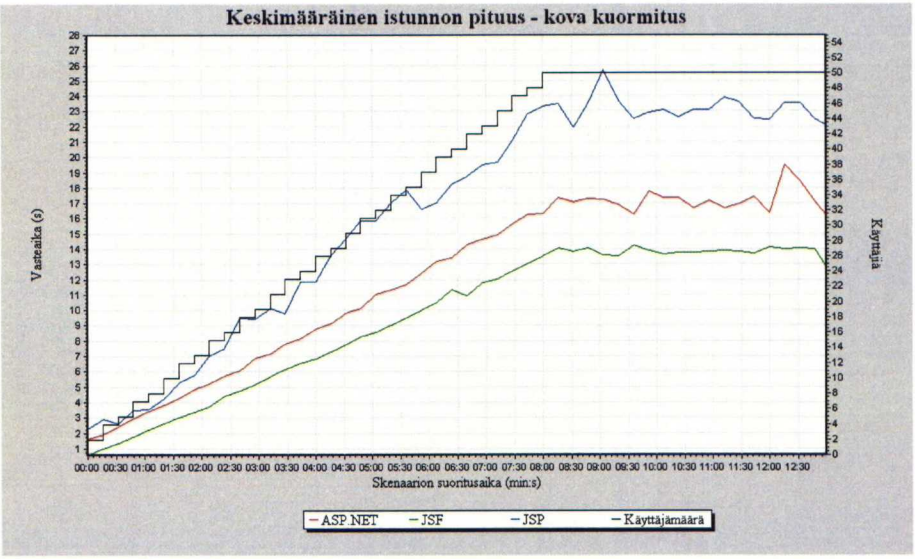
Kuva 37: Toteutusten siirtokyky kohtalaisella kuormituksella.

Taulukossa 1 on esitetty prototyyppien ja JSP-toteutuksen keskimääräiset sivukoot absoluuttisina tavuarvoina sekä sivujen kokojen suhdeluvut. Sivujen koot mitattiin kirjautumalla järjestelmiin testeissä käytettyjen virtuaalikäyttäjien tunnuksilla. Mitattu raporttisivu sisälsi kolmen kustannuspaikan tiedot. Taulukosta nähdään, että ASP.NET- ja JSF-prototyypin aloitusivu ja pääsivu olivat lähes samankokoiset. JSP-toteutuksen sivut olivat komponenttiteknologioiden sivuja pienempiä, ero johtuu pitkälti siitä, että komponenttiteknologiat sisällyttävät sivuihin komponentteihin ja tilanhallintaan liittyviä tunnistetietoja. JSF-prototyypin raporttisivu oli huomattavasti suurempi kuin ASP.NET-prototyypin tai JSP-toteutuksen. Ero johtui siitä, että JSF laittoi jokaiselle raportin solulle oman CSS-tyylimäärityksen.

Taulukko 1: Prototyyppien ja JSP-toteutuksen absoluuttiset ja suhteelliset sivukoot.

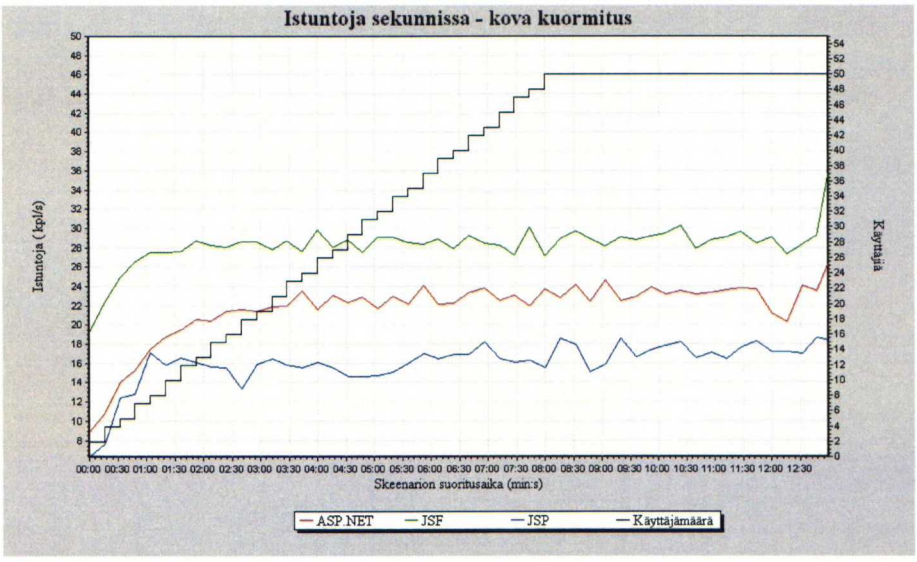
	Aloitussivu	**Pääsivu**	**Raporttisivu**	**Raporttisivu***
ASP.NET	6 512 B	8 334 B	20 577 B	89 019 B
JSF	6 661 B	8 308 B	99 337 B	335 544 B
JSP	3 983 B	7 154 B	18 380 B	69 654 B
ASP.NET / JSP	1,63	1,16	1,12	1,28
JSF / JSP	1,67	1,16	5,40	4,82
ASP.NET / JSF	1,02	1,00	4,83	3,77
* = Raporttisivu kun yksityiskohtaiset rivit ovat valittuna.				

Kuvassa 38 on esitetty keskimääräinen istunnon pituus skenaarion suoritusajan funktiona kun kuormitus suoritettiin ilman käyttäjän harkinta-aikaa simuloivia viiveitä. Vasteajat kasvoivat lineaarisesti suhteessa käyttäjien määrään. JSF-prototyyppi suoriutui parhaiten. ASP.NET-prototyyppi oli suorituskyvyltään lähempänä JSF-prototyyppiä kuin JSP-toteutusta. Kuormituksen tasaisessa vaiheessa JSF-prototyypin vasteaika vaihteli vähiten ja JSP-toteutuksen eniten.



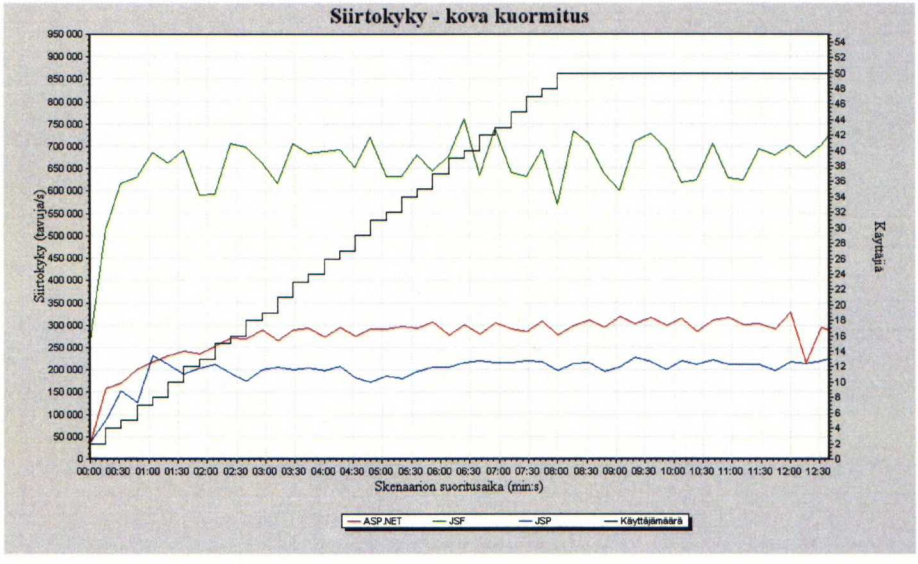
Kuva 38: Keskimääräinen istunnon pituus kovalla kuormituksella.

Kovalla kuormituksella istuntojen määrässä sekuntia kohden JSF-prototyyppi otti kärkipaikan, ASP.NET-prototyyppi seurasi melko lähellä ja JSP-toteutus suoriutui heikoiten (kts. kuva 39). Vaikka käyttäjien määrää lisättiin lineaarisesti, niin istuntojen määrä sekuntia kohden ei kasvanut. Pullonkaulaksi osoittautui tietokantapalvelin, sen prosessorit olivat ylikuormitettuina, mikä aiheutti testeissä sivupyyntöjen jonoutumisen. JSP-toteutus kuormitti tietokantaa eniten, sillä sen suorituskyky alkoi hidastua jo 12 virtuaalikäyttäjän kuormalla. JSF-prototyypin suorituskyky alkoi vakiintua noin 16 käyttäjän kuormalla ja ASP.NET-prototyypin 25 käyttäjän kohdalla.



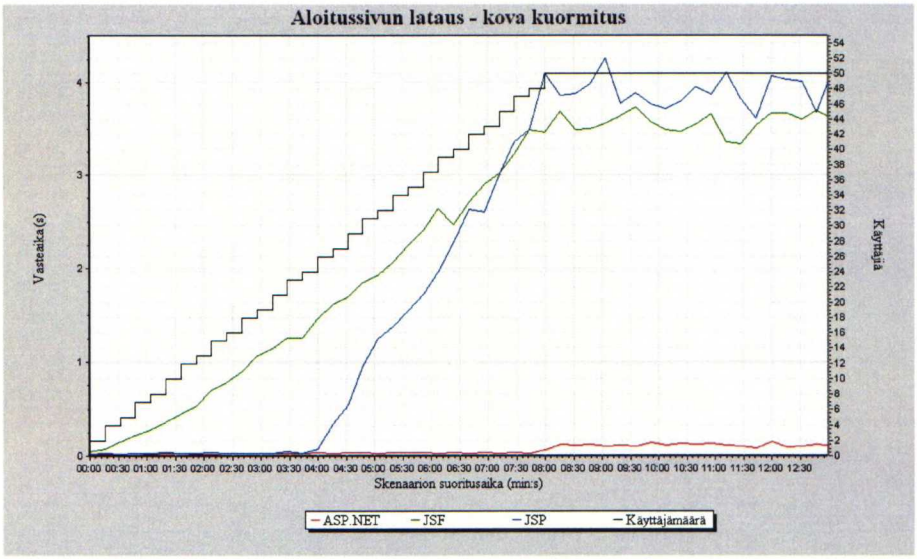
Kuva 39: Suorituskyky kovalla kuormituksella.

Kuvassa 40 on esitetty toteutusten siirtokyky kovalla kuormituksella. Vaikka jokaisen toteutuksen siirtokyky on suurempi kuin kohtalaisella kuormituksella, niin tietokantapalvelimen aiheuttama pullonkaula näkyy kuvassa silti selvästi, sillä jokaisen toteutuksen siirtokyky alkaa vakiintua kun käyttäjien määrä nousee yli kahdeksan. JSF-prototyypin siirtokyky oli melkoisen purskeista, ASP.NET-prototyypin ja JSP-toteutuksen siirtokyvyt oskilloivat huomattavasti vähemmän. JSF-prototyypin suorituskyvyn heilahdukset näkyvät siirtokyvyssä voimakkaammin, koska JSF-sivut olivat kooltaan isompia, kuten taulukossa 1 jo aiemmin esitettiin.



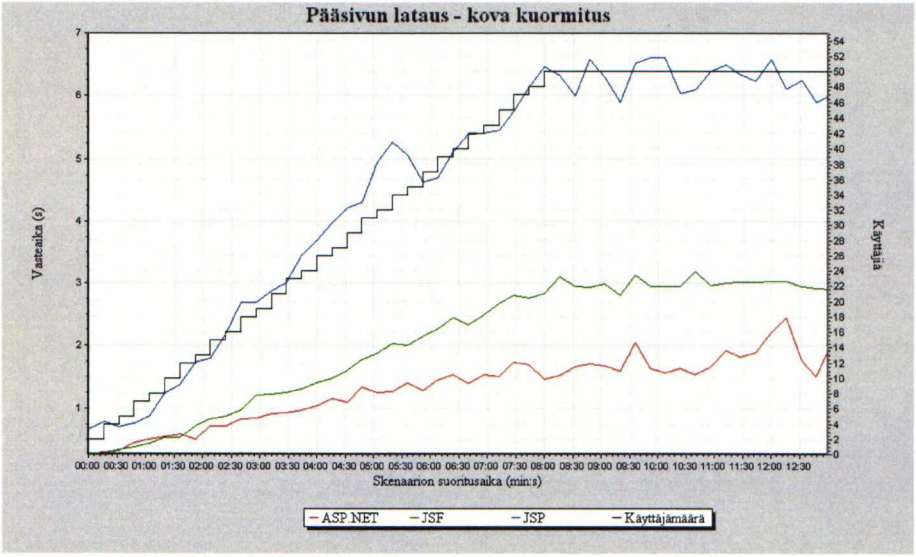
Kuva 40: Toteutusten siirtokyky kovalla kuormituksella.

Aloitussivun latausajassa kovalla kuormituksella ASP.NET-prototyyppi oli aivan omassa teholuokassa (kts. kuva 41) eikä käyttäjämäärällä ollut merkittävää vaikutusta suorituskykyyn. ASP.NET-prototyypin hyvä vasteaika selittyy osin sillä, ettei aloitussivulle ladata tietoa kannasta, vaan se tulee suurimman osan ajasta oliokätkömuistista. JSF-prototyyppi hyödyntää myös oliokätkömuistia, mutta noin 20 käyttäjän kohdalla vasteaika alkaa kasvaa lineaarisesti. Syynä voi olla se, että aloitussivua pyydettäessä JSF-ympäristö alustaa avustavat pavut ja istuntoon liittyvät tietorakenteet. JSP-toteutus pystyi tarjoamaan 50 käyttäjän kuormituksella aloitussivun keskimäärin alle neljässä sekunnissa, joka on vielä hyväksyttävä aika järjestelmän ollessa erittäin suuren kuormituksen alla.



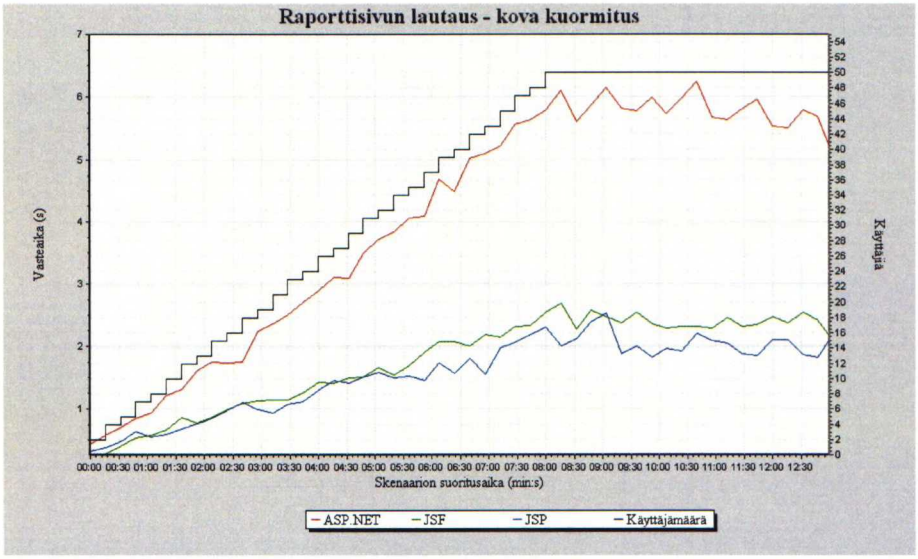
Kuva 41: Aloitussivun latausaika kovalla kuormituksella

Pääsivun latausajat on esitetty kuvassa 42. ASP.NET-prototyyppi suoritui parhaiten keskimääräisen latausajan ollessa alle kaksi sekuntia. JSF-prototyyypi pääsi alle kolmeen sekuntiin ja JSF-toteutus alle seitsemään sekuntiin. Pääsivua ladattaessa tietokannasta haetaan listaus niistä kustannuspaikoista, joihin käyttäjällä on oikeus. Lista on kuitenkin niin lyhyt, ettei pääsivun lataaminen kuormita tietokantapalvelinta merkittävästi. Vasteajat kasvoivat sekä prototyyppien että JSP-toteutuksen osalta lineaarisesti käyttäjien määrän suhteen ja pysyivät lähes vakioina tasaisella kuormituksella.



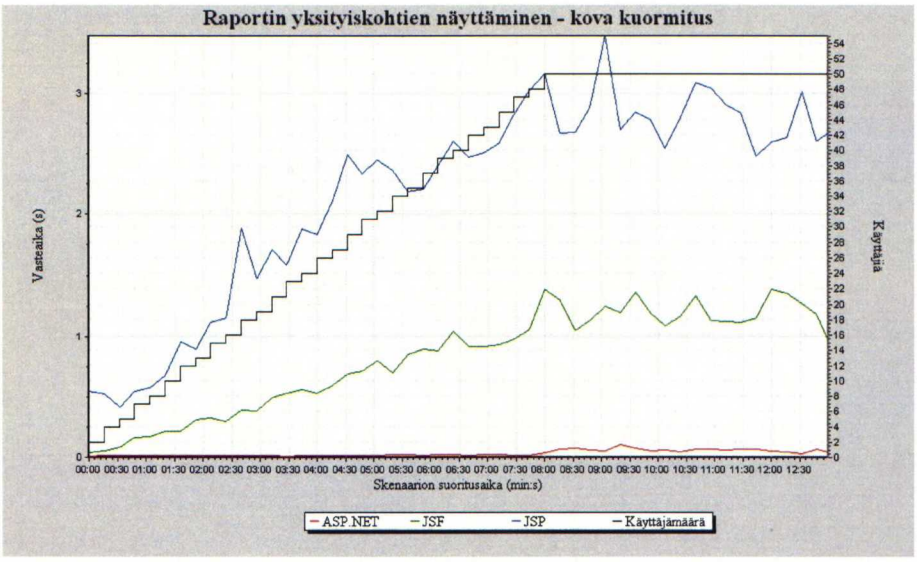
Kuva 42: Pääsivun latausaika kovalla kuormituksella.

Raporttisivun latausajat kovalla kuormituksella on esitetty kuvassa 43. JSF-prototyyppi ja JSP-toteutus suoriutuivat lähes yhtä hyvin JSP-toteutuksen ollessa vain vähän hitaampi. ASP.NET-prototyyppi hidastui lineaarisesti suhteessa käyttäjien määrään. Ero JSP-toteutukseen oli tasaisen kuormituksen vaiheessa yli kolme sekuntia. ASP.NET-prototyyppi pärjäsi kuitenkin JSP-toteutusta paremmin raporttisivun näyttämisessä testeissä, joissa kuormitus oli kohtalaista. JSP-toteutuksen ja JSF-prototyypin vasteajat ovat niin lähellä toisiaan, että niillä on oltava jokin yhteinen tekijä. Raporttisivun latauksessa tietoa haetaan paljon tietokannasta, joten vasteaika riippunee Java-sovelluspalvelimen kyvystä kommunikoida tietokantapalvelimen kanssa. Sekä JSP-toteutus että JSF-prototyyppi käyttivät samaa sovelluspalvelinta ja tietokanta-ajureita.



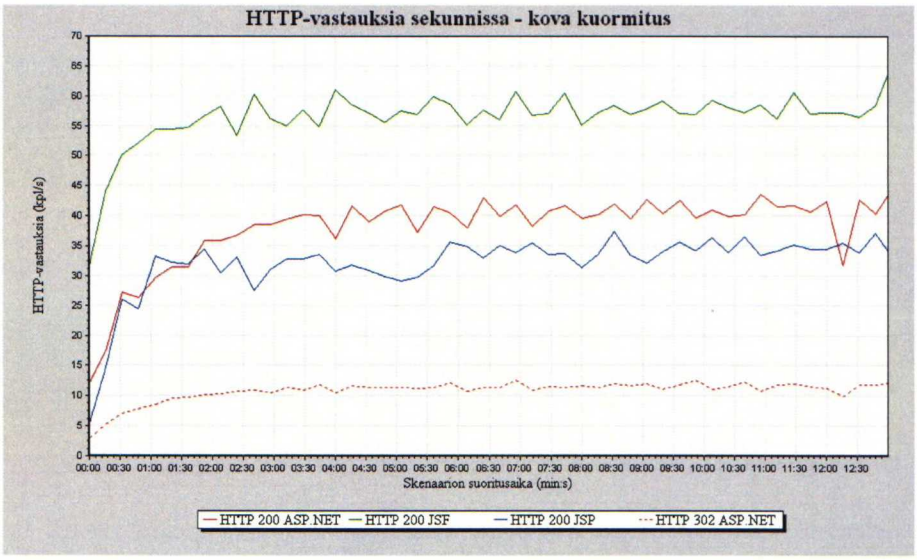
Kuva 43: Raporttisivun latausaika kovalla kuormituksella.

Raportin yksityiskohtaisten rivien näyttäminen (kts. kuva 44) onnistui prototyyppien osalta ilman tietokantahakua, koska raportti löytyi oliokätkömuistista. JSP-toteutus sitä vastoin haki tiedot tietokannasta, mikä näkyy selkeästi vasteajoissa. ASP.NET-prototyypin vasteaika on hyvin lyhyt vaikka itse sivu ei tule kätkömuistista, lyhyt vasteaika toistui kaikilla mittauskerroilla. ASP.NET-prototyypin erittäin nopea vasteaika voi selittyä sillä, että ASP.NET-sivusta löytyy käännetty versio, jota ajonaikainen ympäristö osaa hyödyntää tehokkaasti. Tähän kun lisätään vielä Microsoftin web-palvelimen hyvä yhteistoiminta käyttöjärjestelmän verkkokerroksen kanssa, niin mittaustulokset eivät vaikuta epärealistisilta. Vastaavanlaisia tuloksia saatiin ASP.NET-prototyypille myös aiempana esitellyissä aloitussivua koskevissa mittauksissa.



Kuva 44: Raportin yksityiskohtien näyttämiseen kulunut aika kovalla kuormituksella.

Kuvassa 45 on esitetty HTTP-vastauksien määrä kun toteutuksia kuormitettiin kovalla kuormalla. Tietokantapalvelimen aiheuttama pullonkaula näkyy kuvassa hyvin, koska vastauksien määrä nousee aluksi jyrkästi, mutta vakiintuu sitten tietylle tasolle. Kuvasta näkee selvästi, että komponenttiteknologiat kommunikoivat enemmän web-palvelimen kanssa kuin JSP-toteutus, vaikka suorituskykyerot otetaan huomioon. ASP.NET-ympäristö ilmoitti sivusiirtymistä selaimelle HTTP 302-ilmoituskoodilla.



Kuva 45: HTTP-vastauksien lukumäärä aikayksikköä kohden.

3.4 Ylläpitoprosessi

3.4.1 Ylläpidettävyys

Käyttöliittymäkomponenttiteknologiat helpottavat web-sovellusten ylläpitoa, koska sekä ASP.NET että JavaServer Faces tukevat käyttöliittymien määritysten erottamista itse käyttöliittymälogiikasta. Jos käyttöliittymään pitää tehdä visuaalisia muutoksia, niin ne voidaan antaa web-suunnittelijan vastuulle, samoin web-sovelluskehittäjä voi muuttaa taustalla olevaa ohjelmakoodia ilman, että sillä on vaikutusta itse käyttöliittymän ulkoasuun. Vanhempien toisen sukupolven web-teknologioiden kuten ASP:n ja JSP:n sivumääritykset sisältävät tyypillisesti sekä sivun ulkoasuun vaikuttavia elementtejä että sivun käyttöliittymän hallitsemiseen tarvittavaa ohjelmakoodia, jolloin sivuihin muuttamien on riskialttiimpaa. Sivujen ulkoasun muokkaaminen komponenttiteknologioissa on helpompaa, koska esimerkiksi komponenttien paikkoja voidaan vaihtaa vain muuttamalla niiden XY-koordinaatteja, toisen sukupolven web-teknologioissa sisällön paikan vaihtaminen vaatii kuvauskielisten elementtien muokkausta. JavaServer Faces -sovelluksen sivujen välisiä siirtymiä on helpompi hallita kuin ASP.NET-sovelluksen, sillä muutokset voidaan tehdä suurelta osin vain navigointijärjestelmän konfiguraatio-tiedostoa muokkaamalla. ASP.NET-sovelluksissa sivujen väliset siirtymät on toteutettava joko staattisilla hyperlinkeillä tai dynaamisesti käyttöliittymän ohjelma-logiikassa. Jos navigointi tapahtuu dynaamisesti ohjelmakoodissa, on sovellus käännettävä uudelleen, mikäli navigointia halutaan muuttaa. ASP- ja JSP-sivuilla sivusiirtymät on määritelty yleensä sivujen määrittelydokumentissa, joten navigoinnin muuttaminen on helpompaa kuin ASP.NETissä mutta työläämpää kuin JSF:n tapauksessa.

3.4.2 Uudelleenkäytettävyys

Käyttöliittymäkomponentteja voidaan rakentaa tiettyä tarkoitusta varten tai sitten yleiskäyttöisiksi. Komponenttiteknologioiden mukana tulevat peruskomponentit ovat hyvin yleiskäyttöisiä ja ne soveltuvat käytettäviksi lähes kaikissa web-sovelluksissa. Komponenttien uudelleenkäytettävyyteen vaikuttaa paljon, miten ne on toteutettu. Jos uudelleenkäytettävyys on otettu huomioon jo komponenttia suunniteltaessa, on sen uudelleenkäyttö yleensä helpompaa. Kuten ensimmäisessä luvussa mainittiin, ASP.NET tukee peruskomponenttien lisäksi käyttäjäkomponentteja ja räätälöityjä komponentteja. Käyttäjäkomponenttien rakentaminen kehitysympäristöllä on helppoa, sillä se ei eroa juurikaan ASP.NET-sivun rakentamisesta. Käyttäjäkomponentit voidaan nähdä itsenäisinä ASP.NET-sivun osina. Käyttäjäkomponentit ovat uudelleenkäytettävissä helposti saman sovelluksen sisällä, mutta niiden käyttäminen muissa web-sovelluksissa vaatii sekä määritelmätiedostojen että lähdekooditiedostojen kopioimisen. ASP.NETin räätälöidyt komponentit sitä vastoin voidaan kääntää DLL-tiedostoiksi, jotka sisältävät kaiken komponentit levittämiseen tarvittavan tiedon. Mukautettujen JSF-komponenttien rakentaminen on vaikeampaa kuin niiden käyttäminen. Hyvin rakennettu mukautettu JSF-komponentti on teoriassa uudelleenkäytettävämpi kuin räätälöity ASP.NET-komponentti, koska JSF-komponenttia ei ole sidottu pelkästään HTML-kieleen, vaan samaa komponenttia voidaan näyttää erityyppisille selainlaitteille. Käytännössä JSF-komponentin uudelleenkäyttö voi olla vaikeaa tai jopa mahdotonta eri kehitysympäristöissä, koska komponenteilta puuttuu standardoitu formaatti, jota kaikki kehitysympäristöt ymmärtäisivät. Samaa kehitysympäristöä käytettäessä mukautettujen JSF-komponenttien uudelleenkäyttö on pääsääntöisesti vaivatonta. Komponenttiteknologiat tarjoavat vanhempiin web-teknologihin verrattuna merkittävää uudelleenkäytettävyyttä, esimerkiksi JSP-sivujen rakentamisessa uudelleenkäyttö on saavutettavissa vain sivumäärityksiä kopioimalla tai elementtikirjastoja käyttämällä.

3.5 Vertailukohtien pisteytys

Taulukossa 2 on pisteytetty tässä luvussa käsitellyt vertailukohdat. Pisteytyksessä on käytetty asteikkoa väliltä yksi ja viisi, missä yksi tarkoittaa heikkoa, kaksi tyydyttävää, kolme erittäin tyydyttävää, neljä hyvää ja viisi erinomaista. Pisteet on annettu tämän työn yhteydessä saatujen kokemusten pohjalta sekä hyödyntämällä aiempaa kokemusta JSP-kehityksestä. Kaikkia vertailukohtia ei voitu pisteyttää JSP:n osalta - nämä joko koskivat käyttöliittymäkomponentteja tai aiheita, jotka JSP:n suhteen on rajattu tämän työn ulkopuolelle.

Taulukko 2: Teknologioiden pistemääräinen vertailu.

Vertailukohta	ASP.NET	JSF	JSP
Kehitystyön tuottavuus työkalut huomioiden.	4	3	2
Teknologian kypsyys.	4	2	4
Arkkitehtuuriin laatu kompleksisuus huomioiden.	4	3	2
Teknologian omaksuttavuus.	4	2	3
Teknologian joustavuus.	3	5	3
Käyttöliittymäkomponenttien helppokäyttöisyys.	4	2	-
Käyttöliittymäkomponenttien monipuolisuus.	4	3	-
Kolmansien osapuolten käyttöliittymä-komponenttitarjonta.	5	2	-
Markkinoilla olevat kehitystyökalut	5	3	-
Selainriippumattomuus.	3	3	-
Alustariippumattomuus.	2	4	4
Tuki mobiililaitteille.	2	3	-
Suorituskyky tehtyjen mittauksien perusteella.	4	4	3
Ylläpidettävyys.	4	3	2
Uudelleenkäytettävyys.	4	3	1

3.6 Yhteenveto

Tässä luvussa vertailtiin ASP.NET- ja JavaServer Faces -komponenttiteknologioita toisiinsa sekä vanhempaan toisen sukupolven JavaServer Pages -teknologiaan. Vertailu suoritettiin kehitys-, tuotanto- ja ylläpitoprosessin näkökulmista.

Kehitysvaiheessa arvioitavana oli kehitystyön tuottavuus, teknologioiden kypsyys, kompleksisuus, selainriippumattomuus ja tuki mobiililaitteille. Komponenttiteknologioita käytettäessä kehitystyön todettiin olevan tuottavuudeltaan vanhempia webteknologioita parempi erityisesti kun apuna ovat tehokkaat kehitysympäristöt. ASP.NETin todettiin olevan teknologialtaan kypsempi, sillä se on ollut pidempään markkinoilla. JSF on arkkitehtuuriltaan monimutkaisempi, koska se tukee komponenttien renderöinnin ulkoistamista. Molemmat tutkitut komponenttiteknologiat tuottivat uusissa selaimissa hyvin toimivaa kuvauskieltä. JSF soveltuu arkkitehtuuriltaan paremmin käytettäväksi mobiililaitteiden kanssa, sillä ASP.NET-mobiilikehitys vaatii erityisten mobiilikomponenttien käyttämistä.

Tuotantonäkökulman vertailu tehtiin ajamalla suorituskykymittauksia työn yhteydessä rakennetuille prototyypeille sekä vanhemmalle JSP-toteutukselle. Suorituskykymittaukset pyrkivät simuloimaan mahdollisimman todenmukaisia käyttötilanteita. Testeissä mitattiin istuntojen pituutta, sivupyyntöjen vasteaikoja ja sovellusten siirtokykyä sekä kohtalaisella että kovalla kuormituksella. Mittaustulosten mukaan molemmat prototyypit pärjäsivät hyvin JSP-toteutukselle ollen useimmissa mittauksissa sitä nopeampia. JSF-prototyyppi oli monissa mittauksissa ASP.NET-prototyyppiä nopeampi, mutta sivuilla jotka eivät vaatineet kommunikointia tietokannan kanssa ASP.NET-prototyyppi oli selvästi nopeampi.

Ylläpitoprosessin näkökulmasta tarkastelu rajattiin ylläpidettävyyteen ja uudelleenkäytettävyyteen. Komponenttiteknologiat helpottavat käyttöliittymien ylläpitoa, koska käyttöliittymien määritykset on erotettu niiden ohjelmakoodista. Käyttöliittymäkomponentit ovat uudelleenkäytettäviä, mikäli uudelleenkäyttö on huomioitu jo komponenttien suunnitteluvaiheessa. On mahdollista rakentaa hyvin uudelleenkäytettäviä komponentteja tai sellaisia, jotka sopivat vain yhteen käyttötarkoitukseen.

4 Johtopäätökset

Tässä työssä tutkittiin käyttöliittymäkomponenttipohjaisia ASP.NET ja JavaServer Faces web-teknologioita. Työn yhteydessä rakennettiin molemmilla teknologioilla Nokialla käytössä olevan KP-järjestelmän raportointiosuuden toteuttavat prototyypit. Prototyyppejä verrattiin vanhempaan JavaServer Pages -pohjaiseen toteutukseen mm. suorittamalla mahdollisimman todenmukaisia käyttötilanteita simuloivia suorituskykytestejä.

Työn tarkoituksena oli selvittää mitä lisäarvoa käyttöliittymäkomponentit tuovat web-sovelluskehitykselle. Työn yhteydessä havaittiin, että käyttöliittymäkomponentit nostavat web-käyttöliittymien rakentamisen yhden abstraktiotason verran ylemmäksi. Kehittäjä ei työskentele enää HTTP-protokollan mukaisten sivupyyntöjen ja kuvauskielten parissa, vaan käyttöliittymät muodostetaan käyttöliittymäkomponenteilla ja niiden yhdisteillä. Käyttöliittymäkomponentit aiheuttavat tapahtumia, jotka aktivoivat käyttöliittymälogiikkaa. On huomattava, että käyttöliittymäkomponenttien hyödyntäminen sovelluskehityksessä on merkittävästi helpompaa kuin komponenttien rakentaminen. Käyttöliittymäkomponentteja ei kuitenkaan tarvitse aina rakentaa puhtaalta pöydältä, vaan niitä voi yhdistellä, laajentaa tai hankkia kolmansilta osapuolilta. Käyttöliittymäkomponentit kapseloivat käyttöliittymien toiminnallisuutta ja tekevät täten web-sovelluksista modulaarisempia. Käyttöliittymäkomponentit helpottavat uudelleenkäyttöä, koska samaa komponenttia voidaan käyttää useassa eri web-sovelluksessa. Uudelleenkäytettävyys on kuitenkin huomioitava jo komponentin rakennusvaiheessa. Käyttöliittymäkomponenttiteknologioilla on helpompia luoda dynaamisia käyttöliittymiä kuin vanhemmilla dokumenttipohjaisilla teknologioilla, koska uusia komponentteja on mahdollista lisätä web-sivulle ajonaikaisesti tarpeen vaatiessa web-sovelluksen ohjelmakoodissa.

Prototyyppien käyttöliittymät pystyttiin rakentamaan hyvinkin nopeasti, tuottavuuteen vaikutti paljon käytetyt työkalut. Graafiset kehitysympäristöt nopeuttavat merkittävästi käyttöliittymien rakentamista, ilman niitä kehitystyö on hankalampaa kuin vanhemmilla toisen sukupolven web-teknologioilla.

Tehtyjen mittauksien perusteella ASP.NET ja JavaServer Faces pärjäsivät vanhemmalle JavaServer Pages -teknologialle suorituskyvyssä. Molemmat prototyypit olivat lähes joka tilanteessa JSP-toteutusta tehokkaampia. Saadut mittaustulokset riippuvat hyvin monesta tekijästä kuten käytetyistä laitteistoista,

käyttöjärjestelmistä, web-palvelinohjelmistoista ja tietokantapalvelimesta, mutta tuloksia voi pitää komponenttiteknologioiden osalta vähintään rohkaisevina. Komponenttiteknologiat ovat arkkitehtuureiltaan paljon vanhempia web-teknologioita monimutkaisempia, joten suorituskykyyn tulee kiinnittää huomioita jo varhain suunnitteluvaiheessa. Prototyyppien osalta suorituskykyä optimoitiin tallentamalla usein kysyttyä, mutta harvoin muuttuvaa tietoa kätkömuistiin.

ASP.NET ja JavaServer Faces ovat keskenään kilpailevia teknologioita, joten niistä löytyy paljon yhtäläisyyksiä, mutta myös merkittäviä eroavaisuuksia. ASP.NET on kehittäjän näkökulmasta yksinkertaisempi ja helppokäyttöisempi, mutta samalla rakenteeltaan jäykempi. .NET-kehitystä tekevä sovelluskehittäjä oppii varmasti kohtuullisella panostuksella tekemään ASP.NET-sovelluksia. ASP-sivuja tehneelle kehittäjälle kynnys on astetta korkeampi, koska ASP:n käyttämät skriptikielet ovat huomattavasti ASP.NET-kehitykseen käytettäviä kieliä yksinkertaisempia JavaServer Faces tarjoaa enemmän vaihtoehtoja kuin ASP.NET, mutta on vaikeampi käyttää, kuten tietyt arvostelijatkin ovat todenneet [Ben05-W] [Wag04-W]. JSF:n monimutkaisuus aiheutuu mm. sivumäärityksissä käytetystä määrittelykielestä ja arkkitehtuurista, joka mahdollistaa käyttöliittymäkomponenttien renderöinnin ulkoistamisen. JSP-kehittäjä ei voi suoraan siirtyä tekemään JSF-sivuja, vaan ensin on sisäistettävä vähintään avustavien papujen käyttö ja EL-kielen perusteet.

JSF:n navigointijärjestelmällä on helppo määrittää sivujen väliset ehdolliset ja ehdottomat siirtymät. ASP.NET ei tarjoa mitään sisäänrakennettua navigointijärjestelmää vaan siirtymät pitää toteuttaa joko linkeillä tai ohjelmakoodissa. Microsoftin mukaan ASP.NETin seuraavaan, vuoden 2005 lopulla ilmestyvään, versioon lisätään JSF:n kaltainen navigointijärjestelmä [Evj04]. ASP.NET tarjoaa paremman valikoiman peruskomponentteja ja komponenttien käyttäminen oli Visual Studio.NETissä helpompaa kuin Java Studio Creatorissa. JSF-komponenttien tarjonta oli prototyyppien kehitysajankohtana vähäistä verrattuna ASP.NET-komponenttien tarjontaan. Osa syy tähän oli se, että JSF on teknologiana verraten nuori, mutta komponenttitarjonnan kasvua hidastaa varmasti epäyhteensopivuudet eri JSF-kehitysympäristöjen välillä. JSF-komponenteilta puuttuu yhtenäinen kehitysaikainen standardi, mutta sellainen on Java-yhteisössä suunnitteilla [JSR-276].

Prototyypeistä jatkokehitykseen valittiin ASP.NET-prototyyppi. Valintaa tukivat ASP.NET-prototyypin suorituskykymittauksissa saadut hyväksyttävät testitulokset ja tuki yleisimmille selaimille. Tärkeimmät valintaan johtaneet syyt olivat kuitenkin

ASP.NET-peruskomponenttien monipuolisuus, hyvä viimeistely ja kehitystyön vaivattomuus. Lisäksi kolmansien osapuolien suuri komponenttitarjonta katsottiin eduksi.

Tutkitut käyttöliittymäkomponenttiteknologiat lunastavat annetut lupaukset ja mahdollistavat web-käyttöliittymien rakentamisen aivan uudella tavalla. Jos suuntaus kohti yhä monipuolisempia ja interaktiivisempia web-käyttöliittymiä jatkuu, niin lähitulevaisuudessa komponenttiteknologiat tulevat varmasti valtaamaan markkinoita vanhemmilta web-teknologioilta kun uusia järjestelmiä rakennetaan ja vanhoja päivitetään. Käyttöliittymäkomponentit monipuolistuvat ja tulevat varmasti älykkäämmiksi. Komponentit osaavat tulevaisuudessa automaattisesti määrittää ulkoasunsa selainlaitteelle sopivaksi, jolloin ne pystyvät palvelemaan hyvin erilaisia asiakkaita mobiililaitteista työpöytäkoneisiin. ASP.NET ja JavaServer Faces tuskin syrjäyttävät toisiaan web-kehityksessä, JSF tullee olemaan ykkösvalinta Java-alustalla ja ASP.NET .NET-ympäristöissä.

5 Lähdeluettelo

Kirjallisuuslähteet

[ABBCEGHJ04] Eric Armstrong, Jennifer Ball, Stephanie Bodoff, Debbie Bode Carson, Ian Evans Dale Green, Kim Haase, and Eric Jendrock. *The J2EE™ 1.4 Tutorial*. Sun Microsystems, December 2004.

[DLWM04] Bill Dudney, Jonathan Lehr, Bill Willis, and LeRoy Mattingly. *Mastering JavaServer Faces*. John Wiley & Sons, June 2004.

[Evj04] Bill Evjen. *ASP.NET 2.0 Beta Preview*. Wrox, July 2004.

[GHJV01] Erich Gamma, Richard Helm, Ralph Johnson ja John Vlissides. *Olio-ohjelmointi Suunnittelumallit*. IT Press, 2001.

[HTML-4.01] W3C Recommendation. *HTML 4.01 Specification*. December 1999.

[HTTP-1.1] Network Working Group. *Request for Comments: 2616, Hypertext Transfer Protocol -- HTTP/1.1*. The Internet Society, 1999.

[JSF-1.1] Craig McClanahan, Ed Burns, and Roger Kitain. *JavaServer™ Faces Specification Version 1.1*. Sun Microsystems, May 2004.

[JSF-1.2] Ed Burns and Roger Kitain. *JavaServer™ Faces Specification Version 1.2 Public Review*. Sun Microsystems, April 2005.

[JSP-2.0] Mark Roth and Eduardo Pelegrí-Llopart. *JavaServer Pages™ Specification Version 2.0*. Sun Microsystems, November 2003.

[JSP-2.1] Pierre Delisle, Jan Luehe, and Mark Roth. *JavaServer Pages™ Specification Version 2.1 (Public Review)*. Sun Microsystems, April 2005.

[JSR-276] Java Community Process. *JSR 276: Design-Time Metadata for JavaServer Faces Components*. June 2005.

[JWH03] Stacy Joines, Ruth Willenborg, and Ken Hygh. *Performance Analysis for Java Web Sites*. Addison Wesley, 2003.

[LH02] Jesse Liberty and Dan Hurwitz. *Programming ASP.NET*. O'Reilly, 2002.

[Man04] Kito Mann. *JavaServer Faces in Action*. Manning Publications, November 2004.

[MVBM04] J.D. Meier, Srinath Vasireddy, Ashish Babbar, and Alex Mackman. *Improving .NET Application Performance and Scalability*. Microsoft Corporation, 2004.

[NPBMMNP03] Gregory Nyberg, Robert Patrick, Paul Bauerschmidt, Jeff McDaniel, Raja Mukherjee, Gregory Nyberg, and Robert Patrick. *Mastering BEA WebLogic Server, Best Practices for Building and Deploying J2EE Applications*. Wiley, August 2003.

[RCGHMMNSW01] Simon Robinson, Ollie Cornes, Jay Glynn, Burton Harvey, Craig McQueen, Jerod Moemeka, Christian Nagel, Morgan Skinner, and Karli Watson. *Professional C#*. Wrox Press Ltd., 2001.

[Shi03] Jack Shirazi. *Java Performance Tuning (2nd Edition)*. O'Reilly, January 2003.

[TMQHNL03] David Trowbridge, Dave Mancini, Dave Quick, Gregor Hohpe, James Newkirk, and David Lavigne. *Enterprise Solution Patterns Using Microsoft .Net: Version 2.0 : Patterns & Practices*. Microsoft Press, 2003.

Internet-lähteet

[Base64-W] Base64 encoding.
http://en.wikipedia.org/wiki/Base64

[Ben05-W] Steve Benfield. *JSF: The Ultimate in Flexibility? Or Complexity?* SYS-CON Media, March 2005.
http://java.sys-con.com/read/43952.htm

[CGI-W] The CGI Specification.
http://hoohoo.ncsa.uiuc.edu/cgi/interface.html

[CSS-W] Cascading Style Sheets, W3C.
http://www.w3.org/Style/CSS/

[DotGNU-W] DotGNU Project.
http://www.dotgnu.org

[EDP-W] Event-driven programming.
http://en.wikipedia.org/wiki/Event-driven_programming

[FastCGI-W] FastCGI.
http://en.wikipedia.org/wiki/FastCGI
http://www.fastcgi.com

[Gur04-W] Natty Gur. *Behind the scenes of ASPX files.* The Code Project, January 2004.
http://www.codeproject.com/aspnet/ASPXFILES.asp

[Mono-W] Mono, the open source development platform based on the .NET framework.
http://www.mono-project.com

[MVC-W] Model-View-Controller.
http://en.wikipedia.org/wiki/Model_view_controller

[RAD-W] Rapid Application Development, RAD.
http://en.wikipedia.org/wiki/Rapid_application_development

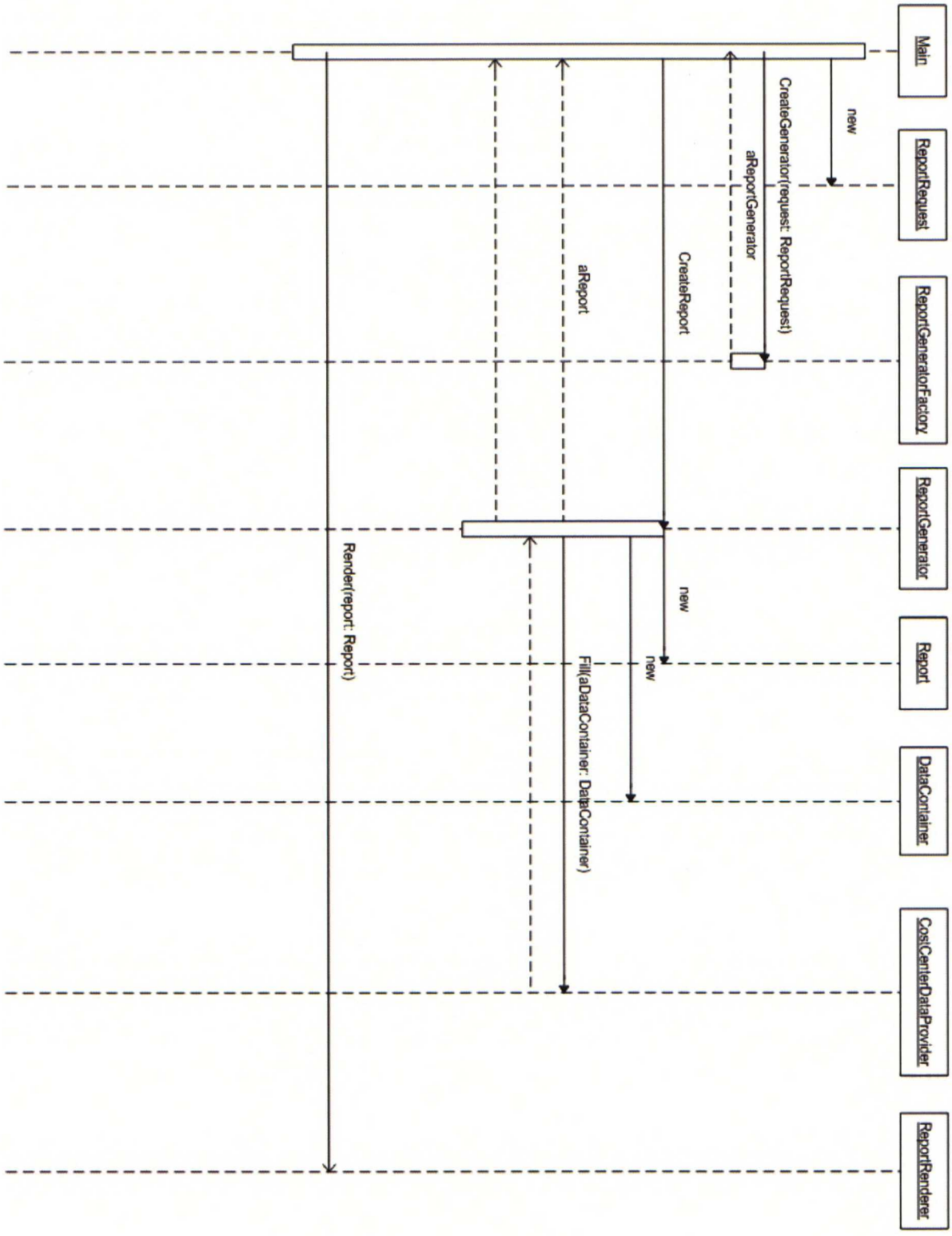
[RFMWD-W] Roadmap for Mobile Web Development with ASP.NET. http://www.asp.net/mobile/mobileroadmap.aspx

[ViewState-W] NET Framework Developer's Guide. *Maintaining State in a Control*. Microsoft MSDN Library. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconMantainingStateInControl.asp

[Wag04-W] Jim Wagner. *Programmers So Far Underwhelmed by JSF.* Internetnews.com, March 2004. http://www.internetnews.com/dev-news/article.php/3322641

[WYSIWYG-W] WYSIWYG, What You See Is What You Get. http://en.wikipedia.org/wiki/WYSIWYG

Liite 1: Raportin generointi ja esittäminen.



Liite 2: Logiikkakerroksen UML-luokkakaavio.